Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications

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

Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications

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 are a key element of telecommunication systems. They are used to recover from packet erasures during large scale content distribution, for instance within the FLUTE/ALC (file transfers) and FECFRAME (continuous real-time media transfers) protocols of 3GPP Multimedia Broadcast and Multicast Services (MBMS). However currently standardized and deployed AL-FEC codes for these protocols (e.g., Raptor(Q) or LDPC-Staircase) are all linear block codes that require the data flow to be segmented into blocks of predefined size. Surprisingly convolutional codes that are based on a sliding encoding window have not yet been considered in spite of their benefits in terms of reduced decoding latency.**

This work analyzes both types of codes in the context of real-time flows. More precisely, it details how to initialize block and convolutional AL-FEC codes to comply with real-time constraints and introduces the "decoding beyond maximum latency" optimization to convolutional codes. Then it compares the erasure recovery performance, the added FEC-related latency, and the decoding throughput of the two codecs. This work highlights the major benefits of convolutional codes for the large scale distribution of real-time flows from all the view points and supports the idea of extending FECFRAME specifications (RFC 6363) to support convolutional FEC codes as well.

***Keywords*—Erasure channel, AL-FEC codes, block codes, convolutional codes, RLC codes**

I. Introduction

Internet and many wireless networks can be regarded as erasure channels, where router congestions, severe packet corruptions that cannot be recovered (e.g., caused by poor reception conditions), or intermittent connectivity are source of packet erasures (i.e., losses). Retransmissions can help improve robustness (e.g., as with TCP), but there are situations where it is not practical: the return channel (used for ACK and/or NACK) does not exist with unidirectional broadcast networks, the extra Round Trip Time (RTT) delay generated by retransmissions may be too large for real-time flows, or scalability issues may prevent the use of a feedback channel (e.g., with multicast/broadcast transmissions to a huge number of receivers).

This is why Application-Level Forward Erasure Correction (AL-FEC) codes have become a key component of large scale content distribution systems, relying on broadcast/multicast technologies to send efficiently the same content to a huge number of receivers. This is the case for file transfer applications, where latency is usually not a major constraint (e.g., the same content can be made available for hours or days, leaving the opportunity for interested users to join the session at their discretion, recover the content and leave). This is also the case for real-time media transfer applications, where latency is an issue (e.g., in case of real-time streaming of sport events in a stadium). The first use-case is typically managed by the **FLUTE/ALC** protocol stack, standardized by IETF as RFC 6726 [1], while the second use-case is managed by the **FECFRAME** protocol stack, standardized by IETF as RFC 6363 [2]. Both protocols are also part of 3GPP (e)MBMS (Multimedia Broadcast and Multicast Services) [3] (and derived) standards, meaning they will soon be deployed on our digital assistants (it is already the case for FLUTE/ALC in certain countries).

However currently standardized AL-FEC codes for FLUTE/ALC and FECFRAME (XOR, Raptor/RaptorQ [4], [5], Reed-Solomon [6] and LDPC-Staircase [7]) are linear block codes: they require the data flow to be first segmented into blocks of predefined size. A major limitation is the added uncompressed latency, caused by the AL-FEC encoding and decoding process, which is penalizing with real-time flows. Surprisingly convolutional codes, based on a sliding encoding window, have not yet been considered for these protocols.

In this work we compare block and convolutional AL-FEC codes for real-time broadcast and multicast distribution (e.g., using FECFRAME), comparing their performance in terms of erasure recovery capabilities, AL-FEC related extra latency, and decoding throughput, using optimized C-language codecs of two representative AL-FEC codes. The contributions of this work are manyfold:

- it explains how to initialize the internal parameters of block and convolutional AL-FEC codes in order to comply with the real-time constraints of the flow(s);
- it introduces the "decoding beyond maximum latency" optimization for convolutional AL-FEC codes that improves their erasure recovery performance at the price of a decoding complexity increase;
- finally, it highlights the major benefits of convolutional codes for the large scale distribution of real-time flows: receivers experiencing good to medium channel quality observe an extra FEC-related latency close to zero, and only very bad receivers will experience an extra latency that approaches that of block codes. On the opposite, with block AL-FEC codes, all receivers

experience the same large latency, no matter their reception quality.

II. Problem Position for Robust Real-Time Flows

Let us consider a real-time flow that has strict timing constraints. This source flow is composed of Application Data Units (ADUs) (we use FECFRAME's terminology [2]) coming from the sending application. ADUs are typically RTP packets containing audio and/or video content. Since each ADU features a maximum validity period, it must be delivered to the receiving application before it expires, which creates constraints on the full transmission chain, from the sender to the receiver. In practice ADUs are potentially of variable size and may contribute to one or more source symbols each. Additionally, FECFRAME prepends a two byte "ADU length" field along with a one byte "flow identifier" field (for the case when several flows are protected together) to each ADU, adds padding such that each augmented be aligned to a multiple of the symbol size.

In order to simplify the analysis **we assume that ADUs are all of the same size and contribute to exactly one source symbol**. We also assume, without loss of generality, that **the network features a constant transmission delay (i.e., there is no jitter) and we ignore UDP/IP processing times** in order to focus on the delays related to AL-FEC coding only. Therefore the only possibility left to control the end-to-end latency are the sender and receiver AL-FEC machinery. This is what we now analyze in detail.

A. The Case of Block AL-FEC Codes

With block AL-FEC codes, the input ADU flow is segmented into a sequence of blocks of appropriate size, and FEC encoding (at a sender) and decoding (at a receiver) are performed independently on a per-block basis. This feature has a major impact on coding/decoding delays, independently of the nature of the code: encoding requires that all the source symbols of the current block be known. Therefore, even if ADUs (source symbols) are immediately sent, repair symbols are necessarily delayed by the block creation time which directly depends on the block size, k, i.e., the number of source symbols in this block. This block creation time also corresponds to the minimum decoding latency any receiver will experience in case of erasures: no repair symbol for the current block can be received before and therefore no erasure recovery can occur before this delay.

A good value for the block size is necessarily a balance between the maximum decoding latency at the receivers (which increases with k) and the desired robustness in front of long erasure bursts (which also increases with k). In practice, the block size k is set to the maximum value made possible by real-time flow requirements, which provides maximum robustness in front of erasure bursts while staying within delay bounds. A downside is that all receivers, even those experiencing excellent transmission conditions, are constrained and penalized by this decoding latency: no erased source symbol (and ADU) can be recovered before this uncompressible delay.

B. The Case of Convolutional AL-FEC Codes

On the opposite, a convolutional AL-FEC code associated to a sliding encoding window of fixed size, or a sliding elastic encoding window of variable size, removes this minimum decoding delay. Indeed, repair symbols are generated and sent on-the-fly, at any time, from the source symbols already present in the current encoding window. At the receiver, an erased source symbol can therefore be recovered thanks to the next repair symbol received if the linear system rank permits it. The FEC-related decoding latency essentially depends on the channel erasure loss and applied coding rate, and is usually rather low for most receivers as we will see. Using a sliding encoding window mode is therefore highly beneficial to real-time flows.

III. Configuration of Block and Convolutional Codes with Real-Time Flows

A. Latency Evaluation Methodology

Let us now discuss the optimal configuration of the two types of AL-FEC codes. We will see that if the situation is pretty simple in case of block codes, this is less trivial in case of convolutional codes. We first make a few assumptions:

- the application is assumed to generate fixed size ADUs, equal to $symb\_size$ bytes. We also assume that each ADU contributes to exactly one source symbol of the same size, $symb\_size$ bytes[1];
- the application is assumed to generate a constant bitrate flow, equal to bw bits per second. This is also the ADU bitrate over the network, and we ignore the repair packet transmission bitrate (not used below);
- the end-to-end ADU and repair symbol transmission over the network is assumed to lead to a constant delay (i.e., there is no jitter), to which we can add the UDP/IP datagram processing times on both ends. This delay can be safely ignored since it results in a fixed offset in the delivering time of ADUs to the receiving application, and reducing this offset is out of scope of this article;
- the receiving application requires ADUs to be delivered sequentially, without ADU mis-ordering. Therefore an erased ADU delays all the following ADUs until the erased ADU is recovered or timed-out.

We also add two definitions:

- each ADU of the application flow has a maximum end-to-end latency constraint, from which we can safely remove the constant communication delay (see above). The resulting transport protocol (e.g., within FECFRAME) maximum latency that cumulates the sender plus receiver processing times is equal to $tp\_max\_lat\_in\_sec$ seconds;
- the AL-FEC code rate[2] at the sender is fixed and equal to cr. It is initialized after considering the worst

[1] We ignore the 3 bytes prepended by FECFRAME to each ADU whose impacts is negligible in front of other parameters.

[2] The code rate is the ratio between the number of source symbols to the total number of source plus repair symbols.

receiver experiencing the highest loss rate that should be supported in the session;

From the above assumptions we see that ADUs are generated at fixed time intervals, and each ADU is immediately sent to the receiver(s). Therefore it forms a convenient time scale, independent of the actual bw and $symb\_size$ values. Therefore we can **measure times in a virtual scale, each ADU generation/transmission corresponding to a "tick" in this virtual time scale**.

It follows that we can define a transport protocol added latency, or tp_added_lat, using this time scale, as the difference between the time when an ADU is expected at a receiver, assuming no erasure occurred, and the time it is actually delivered to the receiving application. When there is no packet erasure, a received ADU can be delivered to the receiving application immediately and features a $tp\_added\_lat = 0$ tick. But an erased ADU is delivered to the receiving application after a certain delay that depends on the type of AL-FEC used, and $tp\_added\_lat > 0$ ticks. This delay can also be infinite in case it cannot be recovered: when this happens we just remove this ADU from the mean calculations.

Minimizing the average and/or maximum transport protocol added latency while keeping the highest possible erasure recovery performance is therefore a key objective of our work. We will see in section V how to evaluate these added latencies through simulations.

Note on the Transport vs. Application Notions of Latency: The added latency as seen by the transport protocol is different from the added latency as seen by the receiving application. Indeed, a video player is obliged to shift ADU playout by the maximum added latency seen so far (e.g., by using a playout buffer of an adequate size), since video content must be consumed in a regular way. This will create differences between the latency statistics obtained in this work and measured at the output of the transport protocol, and the actual latency statistics calculated within the application. In this work we only focus on the transport protocol added latency since our work wants to be independent of the semantic of the transported data flows.

Common	
ADU	Application Data Unit, assumed to contribute to exactly one source symbol in this work
$symb\_size$	symbol size, assumed fixed (in bytes)
bw	application flow bandwidth, assumed fixed (in bps)
$tp\_max\_lat\_in\_sec$	transport protocol maximum latency (in seconds)
tp_max_lat	corresponding transport protocol max. latency (in ticks)
tp_added_lat	transport protocol added latency (in ticks)
cr	AL-FEC coding rate
plr	packet loss rate on the erasure channel
Block codes	
k	block size (in symbols)
Convolutional codes	
ew_size	maximum encoding window size at a sender (in symbols)
dw_size	maximum decoding window size at a receiver (in symbols), or nb of source symbols in L.S. that didn't time-out yet
ls_size	max. linear system size (width) at a receiver (in symbols)

TABLE I. TERMINOLOGY AND NOTATIONS USED IN THIS WORK.

B. The Case of Block AL-FEC Codes

With block codes, the $tp\_max\_lat\_in\_sec$ seconds latency directly maps to a maximum source block size, k:

$$k = tp\_max\_lat = \frac{tp\_max\_lat\_in\_sec * bw}{8 * symb\_size} \quad (1)$$

Indeed, at a receiver, as soon as there is a single erasure, all ADUs that follow this erased ADU need to wait until the erased ADU has been recovered which cannot take place before receiving the $k+1^{th}$ packet, i.e., the first repair symbol. The added latency is therefore comprised between 0 and k ticks, depending on the packet position in the block.

C. The Case of Convolutional AL-FEC Codes

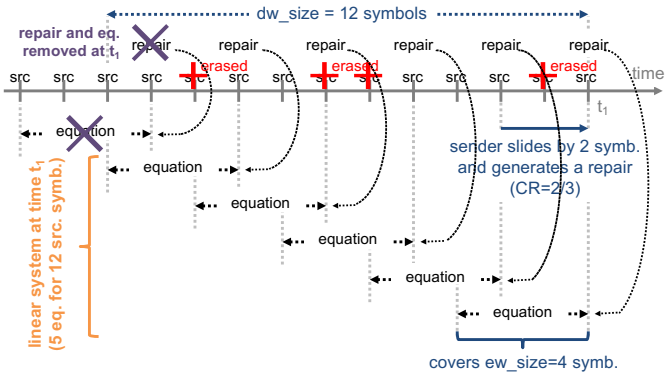


Fig. 1. Sliding window and convolutional code principles, at a receiver, at time t_1 after receiving a new repair symbol (NB: does not comply with Eq. 6).

Two parameters must be considered here (see Fig. 1):

- the maximum encoding window size, equal to ew_size (in symbols): this parameter is used by the sender during repair symbol creation. For instance with RLC codes (Section V-A), each repair symbol is a linear combination of ew_size source symbols.
- the maximum decoding window size at a receiver, equal to dw_size (in symbols): This parameter is the size of the decoding window, i.e., the set of received or erased source symbols that are part of the linear system at a given time. Each time the decoding window slides by a certain number of source symbols to the right, the same number of old source symbols are removed from the left. An equation of this linear system cannot include source symbols that extend beyond this decoding window, and moving the window to the right can trigger the removal of equations from the linear system;

Given our assumption, we can easily calculate the tp_max_lat and corresponding dw_size parameter:

$$dw\_size = tp\_max\_lat = \frac{tp\_max\_lat\_in\_sec * bw}{8 * symb\_size} \quad (2)$$

Given the target code rate cr, the sender will generate a new repair symbol after sliding the encoding window by sl source symbols. Since $cr = \frac{sl}{sl+1}$, it follows that:

$$sl = \frac{cr}{1 - cr} \quad (3)$$

In a given set of dw_size source symbols, there can be a maximum of $dw\_size - ew\_size$ different encoding windows **fully included** in the set. Therefore, from the above results it follows that the maximum number of repairs in a set of of

dw_size source symbols, n_1, equals: $n_1 = \frac{dw\_size - ew\_size}{sl}$ repair symbols, or, more simply:

$$n_1 = (dw\_size - ew\_size) * (\frac{1}{cr} - 1) \quad (4)$$

This is the maximum number of erasures that could be recovered in any set of dw_size source symbols. However this result does not take into account the ew_size parameter, i.e., the individual protection brought by any repair symbol. At an extremum, if $ew\_size = 1$, this protection would be rather low in spite of the high number of repair symbols. Therefore we need to also consider the number of repair symbols that protect a given source symbol in the middle of the set of dw_size source symbols, or n_2. We have: $n_2 = \frac{ew\_size}{sl}$ or more simply:

$$n_2 = ew\_size * (\frac{1}{cr} - 1) \quad (5)$$

This is also the protection of any source symbol in front of an erasure burst.

In practice erasure protection will be determined by the minimum of these two values, n_1 and n_2. We have: $min(n_1, n_2) = (\frac{1}{cr} - 1) * min(dw\_size - ew\_size, ew\_size)$. We reach an optimal protection when:

$$ew\_size = dw\_size/2 \quad (6)$$

IV. Decoding Beyond Maximum Latency

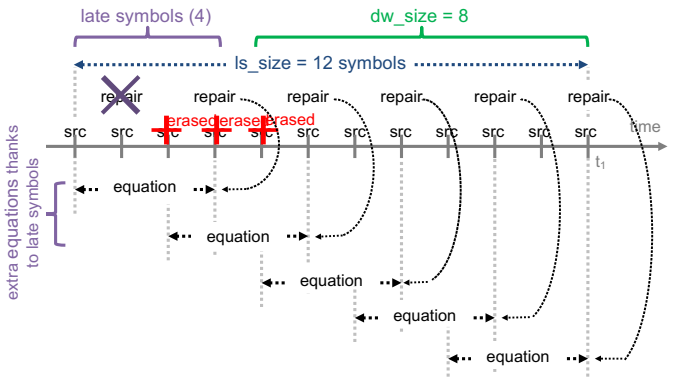


Fig. 2. Late symbols decoding principle, at time t_1 (NB: does not comply with Eq. 6).

In this section we show that we can further improve the decoding performance of convolutional codes without impacting maximum latency, at the cost of extra CPU overhead. The optimization consists, for a receiver, to extend the linear system beyond the decoding window, which means having a ls_size beyond the value defined by Eq. 2 (see Fig. 2):

$$ls\_size > dw\_size \quad (7)$$

It means that source symbols (i.e., ADUs) may be decoded even if their transport protocol added latency exceeds the maximum value permitted by the application. It follows that these source symbols **must not** be delivered to the application and will be dropped once they are no longer needed. However, we show that decoding these late symbols significantly improves the global robustness in bad reception conditions.

The reason is that during a long erasure burst, the initial linear system size, given by dw_size, can turn out to be too small. If a source symbols cannot be recovered, it will impact several equations, as long as it is part of the sender's encoding window. On the opposite, extending the linear system size beyond dw_size offers a better protection against erasure bursts: a source symbol that benefits from a late decoding, even if the associated ADU timed out, may help decoding more recent source symbols within their validity period. Therefore extending the ls_size value at a receiver can help improving robustness even if it triggers additional CPU overhead by decoding symbols that cannot be used by the application[3]. We will show in section V-B that this simple technic is highly beneficial. Additionally, it can be set by a each receiver independently of the source and other receivers, depending on local criteria only (e.g., an energy vs. robustness target).

To summarize, given the application parameters, Equation 2 enables to compute dw_size. Then the sender computes $ew\_size = dw\_size/2$, whereas the receiver can select any value for ls_size such that Eq. 7 holds. We will see that having approximately $ls\_size = 2 * dw\_size$ usually offers a good balance.

V. Erasure Performance Analysis

A. Performance Evaluation Methodology and Choice of Reed-Solomon and RLC Codes

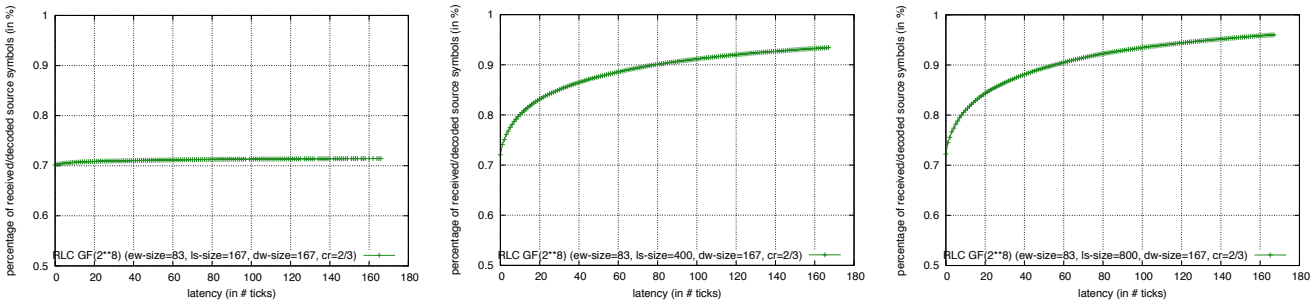
The performance of block and convolutional codes has been assessed through simulated transmissions on a controlled channel, using our OpenFEC (http://openfec.org) AL-FEC implementation and performance evaluation environment. We chose Reed-Solomon as a representative of ideal, MDS block codes, and Random Linear Codes (RLC) as representative of convolutional codes. We only considered Finite Field $GF(2^8)$ and did not try to experiment with sparse variants of RLC in order to only focus on the impacts of the block versus convolutional nature of the codes. We reused our C-language Reed-Solomon codec and derived an RLC codec, both of them relying on the same $GF(2^8)$ library.The `eperftool` evaluation tool of OpenFEC has also been extended to support these codes. Actual AL-FEC encodings and decodings are performed, using C-language codecs, and various low-level parameters enable us to control the simulated channel features. However only memoryless channels are considered in this work. In order to assess decoding throughput, we used a MacbookPro, quadri-core i7/2.5GHz, running MacOS 10.11. Although this powerful laptop is not representative of smartphones, it enables fair comparisons between the two codecs that run exactly in the same environment.

Being compared in the same environment, using the same tools, we have the guaranty that the differences observed in the following sections come from the block versus convolutional AL-FEC nature only.

B. Benefits "Decoding Beyond Maximum Latency"

Let us first analyze the efficiency of the "Decoding Beyond Maximum Latency" strategy (Section IV) in order to find an

[3]Of course the optimum solution, although impractical, consists in having is a linear system that encompasses all the ADUs sent during the session.



(a) RLC CDF when $ls\_size = 167$ (b) RLC CDF when $ls\_size = 400$ (c) RLC CDF when $ls\_size = 800$

Fig. 3. RLC CDF for received or decoded source packet latency distribution, as a function of ls_size. Here ($ew\_size = 83$, $dw\_size = 167$, $CR = 2/3$).

appropriate ls_size value. The evaluation is twofold: from the erasure recovery performance point of view, and from a computational complexity point of view.

We first evaluated the Cumulative Distribution Function (CDF) for RLC codes as a function of the ls_size parameter chosen by a receiver. Since the benefits are mainly visible when approaching the theoretical decoding limits [4], we evaluated the CDF for a channel loss rate equal to 30%, close to the 33.33% maximum theoretical limit made possible by $CR = 2/3$. Figures 3 demonstrate how efficient this approach is: moving from $ls\_size = 167$ (default solution) to $ls\_size = 400$ increases the marginal success probability from 71.43% up to 93.42% which will have major impacts (let's recall here that this success probability only considers "on-time" decodings, packets decoded too late are ignored in the CDF computation). These figures also show that values $ls\_size > 400$ does not significantly increases this success probability (maximum probability reaches 96.02%).

Increasing the ls_size parameter also impacts decoding complexity. Decoding speeds are summarized in table II. We see that $ls\_size = 400$ appears to be a good balance in terms of decoding speed.

ls_size	at 1% loss rate	at 5% loss rate	at 24% loss rate
200	1713.9 Mbps	725.9 Mbps	187.5 Mbps
400	1634.8 Mbps	608.3 Mbps	155.3 Mbps
800	1358.3 Mbps	490.4 Mbps	101.3 Mbps

TABLE II. DECODING SPEEDS FOR RLC CODES AS A FUNCTION OF THE CHANNEL LOSS RATE AND ls_size. HERE ($ew\_size = 83$, $dw\_size = 167$, $CR = 2/3$).

These results validate our choice of choosing $ls\_size = 400$ when the transport added maximum latency $tp\_max\_lat = dw\_size = 167$. **More generally, choosing ls_size roughly equal to twice the dw_size value is considered as a reasonable choice.** This rule will be followed for all the remaining performance evaluations.

C. Transport Protocol Added Latency Performance

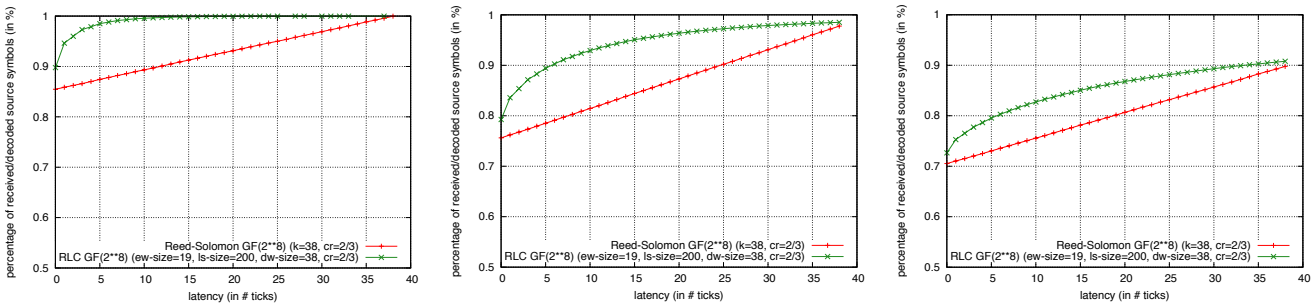
Let us now focus on transport protocol added latency performance. As explained in Section II, Reed-Solomon codes suffer from their block nature when compared to RLC convolutional codes. This is visible in the Cumulative Distribution Function (CDF) figures for received or decoded source packet latency distribution (Figures 4 and 5). No matter whether we consider a small block ($k = 38$) or medium size block ($k = 167$), RLC CDF curve is always and immediately significantly above that of Reed-Solomon. The behavior difference is easily understood when considering the histogram of received or decoded source packet latency distribution in Figures 6 (note that histrograms are truncated in order to improve readability). A large queue distribution remains up to a delay corresponding to the $k = 38$ block size for Reed-Solomon, whereas most ADUs have been received or decoded in less than 10 ticks with RLC.

The same phenomenon is highlighted in a different way in Figure 7, on average across several sessions. This figure also exhibits the residual losses after AL-FEC decoding. We see two different behaviors. With block codes, the first source symbol of a block, if erased, will contribute to a $167 - 0$ ticks latency once recovered, the second source symbol if erased to a $167 - 1$ ticks latency, etc. till the last symbol of the block that will be recovered immediately (we assume here decoding succeeds) contributing to a 0 tick latency. Additionally, symbols can only be sent to the application in order. So if the i^{th} source symbol of a block is erased, all the remaining source symbol of the block that could be received will also be delayed until block decoding took place. With Reed-Solomon codes, a block size $k = 167$ symbols means that each block is on average impacted by at least one erasure, even for very low packet loss rates (1% for instance). Consequently, the curve immediately reaches and stabilizes the average $167/2 = 83.5$ ticks latency when $plr > 0$. The curve is flat. The situation is totally different with convolutional codes as can be seen in the figure, and the progression of the average latency as the plr increases is extremely progressive. The only receivers that will experience a significant FEC-related latency are those that experience very bad channels.

Concerning the residual losses (i.e., ADU erasures that could not be recovered in due time), we see that both codes behave the same, except for very bad channels ($plr = 30\%$ and higher) where Reed-Solomon codes exhibit better results. However we believe that this is not a big issue, and in any case, further increasing the ls_size parameter would help reduce the RLC residual losses as we showed in Section V-B.

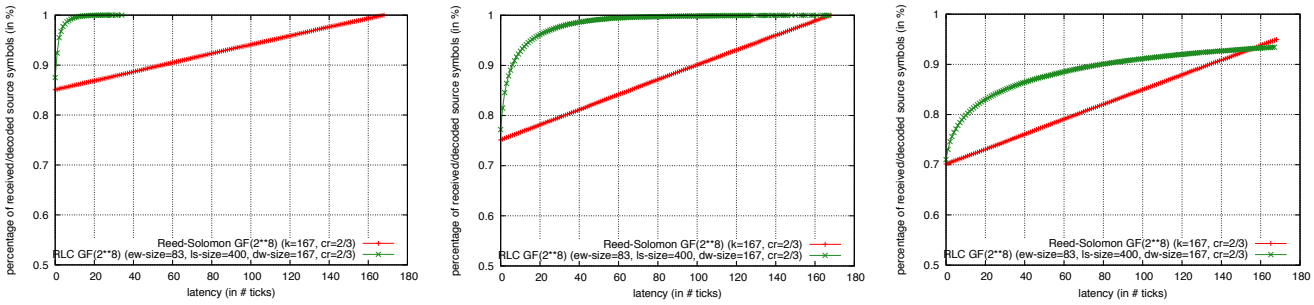
Globally, we can say that RLC codes exhibit an order of magnitude improvement in terms of FEC-added latency compared to Reed-Solomon codes in most situations.

[4] In good reception conditions, all erased source symbols are quickly recovered without having to consider late symbol decoding.



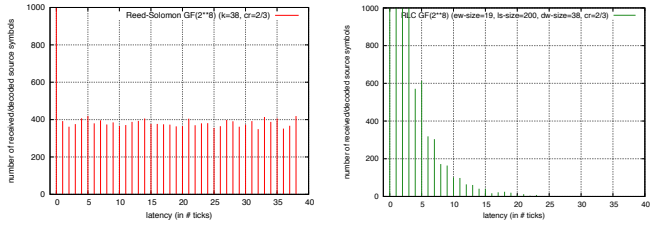
(a) R-S vs. RLC CDF when $loss = 15\%$ (b) R-S vs. RLC CDF when $loss = 25\%$ (c) R-S vs. RLC CDF when $loss = 30\%$

Fig. 4. Reed-Solomon and RLC CDF for received or decoded source packet latency distribution, in case of small blocks (38), for various loss rates of a memory-less channel. Comparable parameters are used for both codes, namely $k = 38$ for Reed-Solomon and ($ew\_size = 19$, $ls\_size = 200$, $dw\_size = 38$) for RLC, with $CR = 2/3$ in all cases. A total of $100,000$ source symbols are sent during tests.



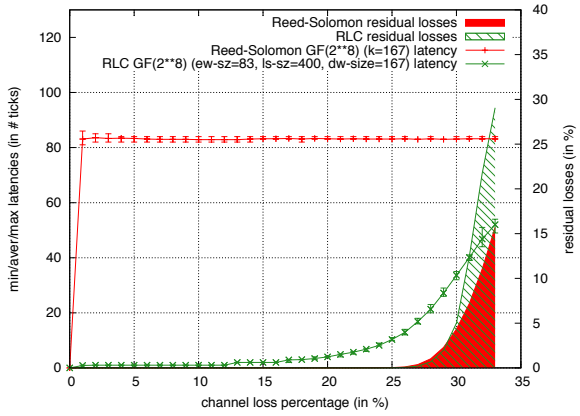
(a) R-S vs. RLC CDF when $loss = 15\%$ (b) R-S vs. RLC CDF when $loss = 25\%$ (c) R-S vs. RLC CDF when $loss = 30\%$

Fig. 5. Reed-Solomon and RLC CDF and associated histograms of received or decoded source packet latency distribution, in case of medium size blocks (167), for various loss rates of a memory-less channel. Comparable parameters are used for both codes, namely $k = 167$ for Reed-Solomon and ($ew\_size = 83$, $ls\_size = 400$, $dw\_size = 167$) for RLC, with $CR = 2/3$ in all cases. A total of $100,000$ source symbols are sent during tests.



(a) R-S histogram when $loss = 15\%$ (truncated, max value is 85472 symbols for latency 0) (b) RLC histogram when $loss = 15\%$ (truncated, max value is 89774 symbols for latency 0)

Fig. 6. R-S and RLC histograms of received or decoded source packet latency distribution, in case of small blocks (38), for loss rate 15%.



(a) RS vs RLC

Fig. 7. Block versus convolutional AL-FEC minimum/average/maximum added latency as a function of the loss rate of a memory-less channel. Comparable parameters are used for both codes, namely $k = 167$ for Reed-Solomon, and ($ew\_size = 83$; $ls\_size = 400$; $dw\_size = 167$) for RLC, with $CR = 2/3$ in all cases.

D. Decoding Speed Performance

Finally we measured the decoding speeds of the RLC and Reed-Solomon codecs[5]. Figures 8 show that if both codecs achieve the same speed when approaching the decoding limit, however RLC exhibits higher speeds in good to medium channel conditions. From this point of view too, there is an incentive to use such convolutional AL-FEC codes as RLC.

[5]Note that previous results [8] already validated the adequacy of this Reed-Solomon codec for lightweight platforms and more generally RLC techniques [9].

VI. Related Works

Although convolutional codes are not yet considered in standardization organizations that rely on AL-FEC codes within upper layers (it's different for lower layers), it has received more attention in the academic community in par-

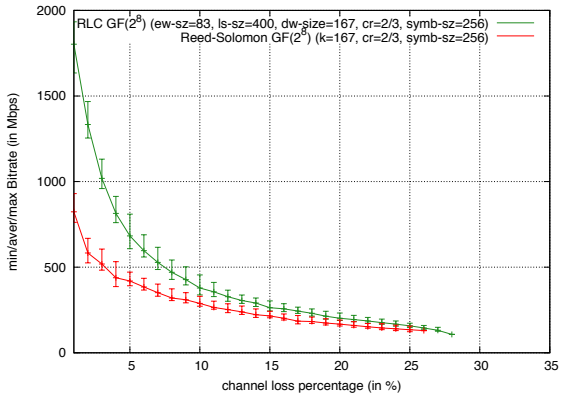


Fig. 8. Reed-Solomon and RLC decoding speed as a function of the loss rate of a memory-less channel. Here $k = 167$ for Reed-Solomon and ($ew\_size = 83$, $ls\_size = 400$, $dw\_size = 167$) for RLC, with $CR = 2/3$ and $symbol\_size = 256$ bytes in all cases.

ticular for network coding use-cases. Random Linear Network Codes (RLNC) are by nature convolutional codes, designed to accommodate in-transit re-coding operations. In this context, their benefit with respect to block codes, has been studied for instance in [10]. Our work shows similarities but differ from several perspectives: we consider broadcast/multicast communications without any feedback channel, while [10] relies on point-to-point, bidirectional communications; we consider end-to-end (or middlebox to middlebox) protection while [10] considers network coding techniques. Additionally, we explain with great details how to configure these codes and introduce a highly efficient "beyond maximum latency decoding" optimization to convolutional codes.

[11] is a recent work on a similar topic. It introduces a detailed theoretic queuing and coding performance analysis to the problem. Being focused essentially on the analytical aspects of low delay codes, it complements well our work that is more practical, introduces an optimization and another approach for performance evaluation.

[12] proposes to use late decoded packets which is similar to our decoding beyond maximum latency. However the work is limited to point-to-point bidirectional communications, where the elastic coding window also evolves according to the acknowledgments obtained from the receiver, and focuses on video flows only. On the opposite, we consider fixed size sliding encoding techniques and carry out performance analyses that are agnostic of the flow nature.

VII. Conclusions and Discussions

This work demonstrates that convolutional AL-FEC codes outperform block codes when dealing with real-time flows, for instance inside FECFRAME: they reduce the FEC-related transport protocol latency by an order of magnitude while keeping similar erasure recovery performance. Considering Reed-Solomon and RLC codes as representative of block and convolutional codes, we also exhibit a decoding speed improvement with good to medium quality channels, which immediately translates into reduced power consumption, an important feature with lightweight terminals. We also explained how to initialize convolutional AL-FEC codes in order to comply with real-time constraints, which is less immediate than with block codes. However, following a few simple step is sufficient. The optimization we proposed, whereby each receiver can, independently of others, decide to extend the maximum size of the linear system, turned out to be extremely efficient in order to reduce the FEC-related latency and residual losses in very bad channel conditions.

These results motivate our work on FECFRAME version 2 [13], an update of RFC 6363 [2] that accommodates convolutional codes. They also prove that RLC codes are a practical solution. We expect to further improve the RLC encoding/decoding speeds with Structured RLC codes (SRLC) [14]. The idea is to replace the dense nature of RLC codes by a structure that benefits from the improved speed of sparse binary linear codes (e.g., LDPC-Staircase codes) in case of large encoding windows, and the almost ideal recovery performance of dense non binary codes (e.g., RLC codes) in case of small encoding windows.

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.

[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, and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery," Oct. 2007, IETF Request for Comments, RFC 5053 (Standards Track).

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] M. Karzand, D. Leith, J. Cloud, and M. Medard, "Fec for lower in-order delivery delay in packet networks," in *arXiv:1509.00167v2*, Sep. 2016.

[12] 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.

[13] V. Roca and A. Begen, "Forward error correction (fec) framework version 2," Oct. 2016, IETF TSVWG work in progress, draft-roca-tsvwg-fecframev2-02.

[14] 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.