MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Rateless Feedback Codes

Sorensen, J.H.; Koike-Akino, T.; Orlik, P.

TR2012-052 July 2012

Abstract

This paper proposes a concept called rateless feedback coding. We redesign the existing LT and Raptor codes, by introducing new degree distributions for the case when a few feedback opportunities are available. We show that incorporating feedback to LT codes can significantly decrease both the coding overhead and the encoding/decoding complexity. Moreover, we show that, at the price of a slight increase in the coding overhead, linear complexity is achieved with Raptor feedback coding.

IEEE International Symposium on Information Theory Proceedings

Rateless Feedback Codes

Jesper H. Sørensen*†, Toshiaki Koike-Akino†, and Philip Orlik†
*Aalborg University, Department of Electronic Systems, Fredrik Bajers Vej 7, 9220 Aalborg, Denmark
†Mitsubishi Electric Research Laboratories (MERL), 201 Broadway, Cambridge, MA 02139, USA
E-mail: jhs@es.aau.dk, {koike, porlik}@merl.com

***Abstract*—This paper proposes a concept called rateless feedback coding. We redesign the existing LT and Raptor codes, by introducing new degree distributions for the case when a few feedback opportunities are available. We show that incorporating feedback to LT codes can significantly decrease both the coding overhead and the encoding/decoding complexity. Moreover, we show that, at the price of a slight increase in the coding overhead, linear complexity is achieved with Raptor feedback coding.**

I. Introduction

Achieving reliable communications over erasure channels has been an active research topic for many years, especially when packet-based communications emerged with the rise of the Internet. When a strong feedback channel is available, the capacity of the erasure channels is easily achieved with well-known automatic repeat-request (ARQ) schemes. The receiver simply feeds back an acknowledgment on each individual symbol, and any unacknowledged symbols are retransmitted. The fact that ARQ makes extensive use of feedback channels makes it difficult to employ this scheme in many practical systems. A rateless coding can achieve reliable communications on erasure channels with no frequent feedback. The so-called LT codes [1] and Raptor codes [2] are such examples. These codes generate a potentially infinite amount of encoded symbols from k information symbols, with an encoding and decoding complexity order of $\mathcal{O}(k \log k)$. A successful decoding is possible when $(1+\epsilon)k$ encoded symbols have been received, where ϵ is a coding overhead.

Now the question is; assuming we have a communication system, where the receiver has m feedback opportunities ($0 < m < k$), what scheme should be applied? While the existing schemes are designed for the extreme cases, $m = k$ and $m = 0$, the more practical assumptions pose an interesting problem. An approach called doped fountain coding is proposed in [3], where the receiver feeds back information on undecoded symbols. This makes the transmitter able to transmit input symbols, which accelerate the decoding process. In [4], another approach called real-time oblivious erasure correcting is proposed. This approach utilizes feedback telling how many of the k input symbols have been decoded. With this information the transmitter chooses a fixed degree for future encoded symbols, which maximizes the probability of decoding new symbols. The same type of feedback is applied in [5], but with the purpose of minimizing the redundancy.

We propose in this paper redesigned versions of LT and Raptor codes, called LT feedback codes and Raptor feedback codes, where any amount of feedback opportunities are taken into account. We focus on more informative feedback than simple acknowledgments to tell the transmitter which information symbols have been recovered, not just how many. This information is used to modify the degree distribution in the encoder. The goal is to decrease the coding overhead especially for short k, in which the conventional rateless codes are inefficient. We will show that the proposed feedback scheme can decrease both the coding overhead and the complexity.

II. Background of Rateless Codes

An overview of standard LT codes is first described here. Assume we wish to transmit data, which is divided into k *input symbols*. From the input symbols, a potentially infinite amount of encoded symbols, called *output symbols*, are generated. Output symbols are XOR combinations of input symbols. The number of input symbols used in the XOR is referred to as the *degree* of the output symbol, and all input symbols contained in an output symbol are called *neighbors* of the output symbol. The output symbols follow a certain degree distribution Ω. The encoding process can be broken down into three steps: 1) Randomly choose a degree d by sampling $\Omega(d)$. 2) Choose uniformly at random d of the k input symbols. 3) Perform XOR of the d chosen input symbols. The resulting symbol is the output symbol. This process can be iterated as many times as needed, which results in a rateless code.

A widely used decoder for LT codes is a belief propagation (BP) decoder whose complexity is low [2]. First, all degree-1 output symbols are identified, which makes it possible to recover their corresponding neighboring input symbols. These are moved to a storage referred to as the *ripple*. Symbols in the ripple are *processed* one by one, which means they are XOR'ed with all output symbols, who have them as neighbors. Once a symbol has been processed, it is removed from the ripple and considered decoded. The processing of symbols in the ripple will potentially reduce the buffered symbols to degree one, in which case the neighboring input symbol is recovered and moved to the ripple. This is called a symbol *release*. Such an iterative decoding process can be explained in two steps: 1) Identify all degree-1 symbols and add the corresponding input symbols to the ripple. 2) Process a symbol from the ripple, remove it afterwards and go to step 1. Decoding succeeds when all input symbols are recovered. If at any point, the ripple size equals zero, decoding has failed.

III. Rateless Feedback Codes

The ripple size is an important parameter in the design of LT codes. This design contains two steps; 1) find a suitable ripple evolution to aim for, and 2) find a degree distribution which achieves that ripple evolution. This section describes such a design for the case where feedback opportunities exist. We first focus on the case of a single feedback located when

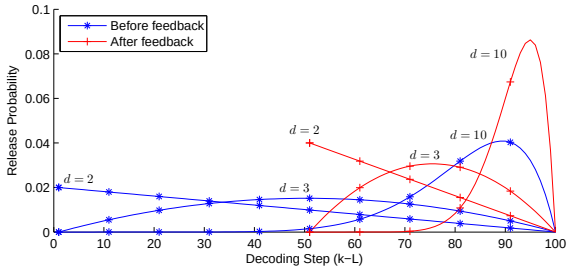

Fig. 1. The release probability as a function of the decoding step.

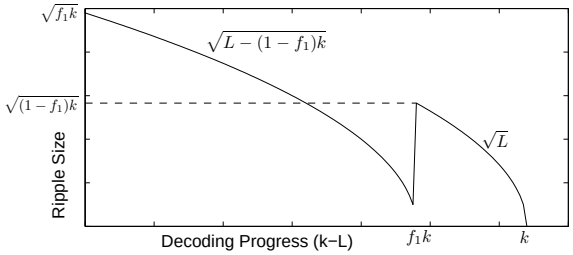

Fig. 2. The proposed ripple evolution for LT feedback codes.

$f_1 k$ symbols have been decoded. It will be later scaled to any amount of feedback opportunities.

A. LT Feedback Codes

The feedback informs the transmitter which input symbols have been decoded. With the information, the encoder excludes all decoded symbols from future encoding. An important implication of this is that the processing of the first $f_1 k$ symbols has no influence on the release of the symbols received after the feedback because no symbol encoded after the feedback has those first $f_1 k$ processed symbols as neighbors. In order to fully understand the benefit from this, we look at a proposition presented in [1]. It expresses the probability, $q(d, L, k)$, that a symbol of degree d is released when L input symbols remain unprocessed, i.e. in the $(k-L)$-th decoding step:

$$q(d,L,k) = \frac{d(d-1)L\prod_{j=0}^{d-3}(k-(L+1)-j)}{\prod_{j=0}^{d-1}(k-j)}, \quad \text{for } d=2,\ldots,k, \text{ and } L=k-d+1,\ldots,1, \tag{1}$$
$$q(1,k,k)=1, \quad q(d,L,k)=0, \text{ for all other } d \text{ and } L.$$

It holds for the traditional LT code without feedback. With feedback, we can divide encoded symbols into two groups; one for symbols received prior to the feedback and one for symbols received after. Symbols received before the feedback follow the release probabilities in (1). Symbols received after the feedback are based on the reduced set of input symbols of size $(1-f_1)k$. Moreover, their releases are independent of the processing of the first $f_1 k$ input symbols. We can therefore find their release probabilities as $q(d, L, (1-f_1)k)$, for $L > f_1 k$.

Fig. 1 shows a plot of $q(d, L, k)$ (before feedback) and $q(d, L, (1-f_1)k)$ (after feedback), with parameters $k = 100$ and $f_1 = 0.5$. It illustrates how releases of symbols received after the feedback are confined to the end of the decoding process. This is useful when designing degree distributions, since it gives more freedom to distribute the releases throughout the decoding process. For example, see the release distributions of low degree symbols after the feedback; they are released with high probability immediately after the feedback. This means that we have the opportunity to give the ripple a boost at an intermediate point in the decoding process. This should be exploited in the design of the LT feedback code.

In [2], it was argued that the ripple size should be kept larger than $c\sqrt{L}$, for some positive constant c. The justification is based on viewing the ripple evolution as a simple random walk, i.e. every time a symbol is processed, the ripple size is either increased or decreased by one with equal probabilities. In such a random walk, the expected distance from the origin after z steps is $\sqrt{z}$. Hence, if L steps remain in the decoding process, the ripple size should be of the order of $\sqrt{L}$ in order to avoid decoding failure. The choice of ripple evolution for the LT feedback code is based on the same reasoning. However, due to the feedback, the ripple evolution can be viewed as two random walks. The first random walk represents decoding until the feedback, i.e. when $(1-f_1)k$ remain undecoded. Hence, we argue that until this point, the ripple size should be equal to $c_1\sqrt{L-(1-f_1)k}$, for a positive constant c_1. The second random walk represents decoding after the feedback, where all remaining symbols must be recovered. Hence, the ripple size should be $c_2\sqrt{L}$, for a positive constant c_2. Fig. 2 illustrates the proposed ripple evolution for $c_1 = c_2 = 1$. The proposed ripple evolution, $R(L)$, is summarized as below:

$$R(L) = \begin{cases} c_1\sqrt{L-(1-f_1)k}, & \text{for } L > (1-f_1)k, \\ c_2\sqrt{L}, & \text{for } L \le (1-f_1)k, \end{cases} \tag{2}$$

for suitable constants c_1 and c_2.

The next step in the design is to find a degree distribution which achieves the proposed ripple evolution (2). Two degree distributions must be found, for the case of a single feedback, one for before the feedback and the other for after. First, the vector $R(L)$ is mapped into another vector, $Q(L)$, which denotes the expected number of releases in the $(k-L)$-th decoding step. We derive this mapping assuming all releases in a single step are unique, i.e. the same input symbol will not be recovered twice within a single step. This is a valid assumption, since the number of releases in a single step is low compared to L. The mapping can be expressed as

$$Q(L) = \begin{cases} \dfrac{L\big(R(L)-R(L+1)+1\big)}{L-\big(R(L+1)-1\big)}, & \text{for } k > L \ge 0, \\ R(L), & \text{for } L = k. \end{cases} \tag{3}$$

Plugging (2) into (3), the desired $Q(L)$ is obtained as

$$Q(L) = \begin{cases} c_1\sqrt{f_1 k}, & \text{for } L = k, \\ \dfrac{L(c_1\sqrt{L-(1-f_1)k}-c_1\sqrt{L-(1-f_1)k+1}+1)}{L-(c_1\sqrt{L-(1-f_1)k+1}-1)}, & \text{for } k > L > k_1', \\ c_2\sqrt{L}, & \text{for } L = k_1', \\ \dfrac{L(c_2\sqrt{L}-c_2\sqrt{L+1}+1)}{L-(c_2\sqrt{L+1}-1)}, & \text{for } k_1' > L \ge 0, \end{cases} \tag{4}$$

TABLE I
DEGREE DISTRIBUTION FOR LT FEEDBACK CODE.

d	$\Omega_1(d)$	$\Omega_2(d)$
1	0.0841	0.1948
2	0.5670	0.2143
3	0.1294	0.1730
4	0.1902	0.1132
5	0.0293	0.0711
6	0	0.0485
7	0	0.0310
8	0	0.0354
10	0	0.0408
13	0	0.0296
15	0	0.0041
17	0	0.0163
19	0	0.0055
22	0	0.0107
25	0	0.0030
27	0	0.0049
30	0	0.0025
31	0	0.0004
32	0	0.0008

TABLE II
DEGREE DISTRIBUTION FOR RAPTOR FEEDBACK CODE.

d	$\Omega_1(d)$	$\Omega_2(d)$
1	0.0854	0.1926
2	0.5618	0.2458
3	0.2224	0.1834
4	0.1304	0.1100
5	0	0.0669
6	0	0.0560
8	0	0.0524
9	0	0.0271
13	0	0.0239
14	0	0.0240
21	0	0.0025
22	0	0.0154

where $k_1' = (1 - f_1)k$. The achieved $Q(L)$ can be expressed as a function of the applied degree distribution, $\Omega(d)$, through (1). This is done for the general case without feedback as

$$Q(L) = \sum_{d=1}^{k-L+1} n\Omega(d)q(d, L, k), \tag{5}$$

where n denotes the number of received symbols in decoding.

For the case with feedback, we have contributions from two different degree distributions, $\Omega_1(d)$ and $\Omega_2(d)$, when $L < (1 - f_1)k$. Correspondingly, n_1 and n_2 symbols have been received prior to decoding. When $\Omega_2(d)$ is applied, the encoder only includes the remaining $(1-f_1)k$ input symbols. Therefore, we have

$$Q(L) = \begin{cases} \displaystyle\sum_{d=1}^{k-L+1} n_1\Omega_1(d)q(d, L, k), & \text{for } L > k_1', \\ \displaystyle\sum_{d=1}^{k_1'-L+1} n_2\Omega_2(d)q(d, L, k_1') + \sum_{d=1}^{f_1 k} n_1\Omega_1(d)q(d, L, k), & \text{for } L \le k_1'. \end{cases} \tag{6}$$

Equations (4) and (6) yield (7) shown at the top of the next page. Here, n_1 and n_2 are the free parameters. Hence, a solution tells us how many symbols we must collect of the different degrees before and after the feedback in order to achieve the desired ripple evolution. Normalizing the solution vectors with n_1 and n_2 provides the degree distributions. Since the matrix in (7) becomes singular for high k, we propose to use the least-squares nonnegative solution for (7) to achieve close to the desired ripple evolution. Table I exemplifies the solution for $k = 128$, $f_1 = 0.75$ and $c_1 = c_2 = 1$. The actual expected ripple evolution achieved with these degree distributions is easily found through the use of (7) and (3). Fig. 3 demonstrates that the least-squares solution closely approaches the proposed evolution.

The design procedure explained above is easily extended to multiple feedback opportunities. Consider m feedback opportunities whose locations are at f_i, for $i = 1, 2, \ldots, m$. The

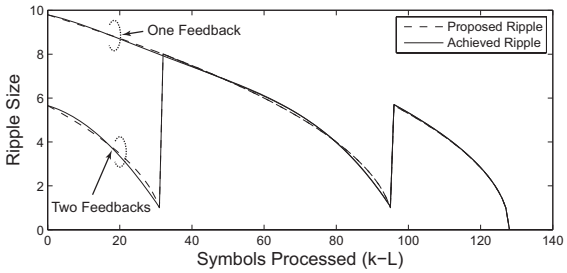

Fig. 3. The achieved ripple evolution compared to the proposed.

decoding can be viewed as $m + 1$ random walks, for which the proposed ripple evolution is written as

$$R(L) = \begin{cases} c_1\sqrt{L - k_1'}, & \text{for } L > k_1', \\ c_i\sqrt{L - k_i'}, & \text{for } k_{i-1}' > L > k_i', \\ c_{m+1}\sqrt{L}, & \text{for } L \le k_{m+1}', \end{cases} \tag{8}$$

where $k_i' = (1 - f_i)k$, with suitable constants c_i for $i = 1, 2, \ldots, m + 1$.

An example of two feedback opportunities for $k = 128$, $m = 2$, $f_1 = 0.25$, $f_2 = 0.75$ and $c_1 = c_2 = c_3 = 1$ is shown in Fig. 3. It also compares what is achieved with the least-squares solution to a generalized version of (7):

$$\begin{bmatrix} H_{1,1} & 0 & 0 & 0 & 0 \\ \vdots & \ddots & 0 & 0 & 0 \\ \vdots & H_{i,j} & H_{i,i} & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ H_{m+1,1} & \cdots & \cdots & \cdots & H_{m+1,m+1} \end{bmatrix} \begin{bmatrix} n_1\Omega_1 \\ \vdots \\ n_i\Omega_i \\ \vdots \\ n_{m+1}\Omega_{m+1} \end{bmatrix} = \begin{bmatrix} Q(k) \\ \vdots \\ \vdots \\ \vdots \\ Q(1) \end{bmatrix}, \tag{9}$$

where, with notations $k_0' = k$ and $k_{m+1}' = 0$,

$$H_{i,i} = \begin{bmatrix} q(1, k_{i-1}', k_{i-1}') & 0 & 0 \\ \vdots & \ddots & 0 \\ q(1, k_i' + 1, k_{i-1}') & \cdots & q(k_{i-1}' - k_i', k_i' + 1, k_{i-1}') \end{bmatrix},$$

$$H_{i,j} = \begin{bmatrix} q(1, k_{i-1}', k_{j-1}') & \cdots & q(k_{i-1}' - k_i', k_{i-1}', k_{j-1}') \\ \vdots & \vdots & \vdots \\ q(1, k_i' + 1, k_{j-1}') & \cdots & q(k_{i-1}' - k_i', k_i' + 1, k_{j-1}') \end{bmatrix}.$$

An important performance metric is the average degree, $\bar{d}$. When no feedback is considered, it is calculated as $\sum_{d=1}^{k} d\,\Omega(d)$. For the proposed LT feedback code, it is calculated using the following equations:

$$\bar{d} = \sum_{i=1}^{m+1} \frac{n_i}{n} \sum_{d=1}^{k_{i-1}' - k_i'} d\,\Omega_i(d), \qquad n = \sum_{i=1}^{m+1} n_i. \tag{10}$$

Fig. 4 shows a comparison of LT feedback codes with increasing number of feedback opportunities and the Robust Soliton distribution (RSD). The RSD proposed in [1] is the *de facto* standard for traditional LT codes. The legend shows the number of feedbacks in parenthesis. The LT feedback code with zero feedbacks, refers to a code using a degree distribution which achieves $R(L) = \sqrt{L}$.

$$\begin{bmatrix} q(1,k,k) & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & & & & \\ \vdots & & 0 & 0 & 0 & 0 \\ q(1,k'+1,k) & \cdots & q(f_1k,k'+1,k) & 0 & 0 & 0 \\ q(1,k',k) & \cdots & q(f_1k,k',k) & q(1,k',k') & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ q(1,1,k) & \cdots & q(f_1k,1,k) & q(1,1,k') & \cdots & q(k',1,k') \end{bmatrix} \begin{bmatrix} n_1\Omega_1(1) \\ \vdots \\ n_1\Omega_1(f_1k) \\ n_2\Omega_2(1) \\ \vdots \\ n_2\Omega_2(k') \end{bmatrix} = \begin{bmatrix} Q(k) \\ \vdots \\ \cdot \\ \cdot \\ \vdots \\ Q(1) \end{bmatrix} \tag{7}$$

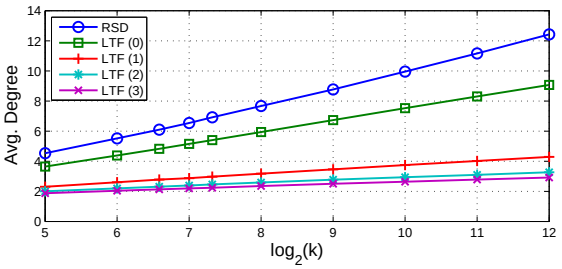

Fig. 4. Average degree of LT feedback codes and traditional LT codes (RSD).

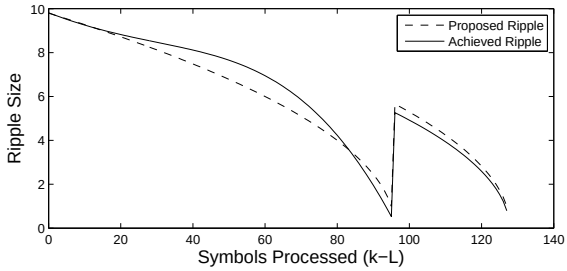

Fig. 5. The ripple evolution achieved in the Raptor feedback code compared to the one proposed for LT feedback codes.

Fig. 4 shows that LT feedback codes decrease the average degree, and thereby computational complexity. It should be noted that as the number of feedbacks increases, $\bar{d}$ as a function of k approaches a constant, which suggests linear complexity. Superlinear complexity is one of the drawbacks of traditional LT codes and we can now conclude that feedback is a way to solve it. However, at small m the average degree still increases with k in the order of $\log(k)$. It is therefore relevant to consider the concept of Raptor coding, which is another approach to ensuring linear complexity [2].

B. Raptor Feedback Codes

Raptor coding is a concatenation of an LT code and a high-rate block code, e.g. low-density parity-check (LDPC) code. During encoding, the block encoder is first applied on the k input symbols. These block-encoded symbols are fed to an LT encoder, which creates a rateless code. Successful decoding is possible whenever the LT decoder has recovered enough symbols for the block decoder to succeed. It was shown in [2] that there exist degree distributions of the LT code to ensure successful decoding after receiving $(1+\epsilon)k$ symbols and has a constant average degree for increasing k. We will now show how our design approach of LT feedback coding can be extended to a Raptor feedback code.

For a given desired $\bar{d}$ in (10), a constraint is easily added to the equation in (7), as shown in the second row from the bottom in (11), which is shown at the top of the next page. Since we can find the least-squares solutions to this system of equations, a weight, $w \gg 1$, has been multiplied in this row, in order to make it more strict. Moreover, in order to avoid the case that the ripple lacks robustness around the feedback point, a constraint is added to ensure that the correct amount of releases is achieved prior to the feedback. This is also a strict constraint and is therefore also multiplied by w. It has been added as the last row in (11).

The least-squares solution for $k = 128$, $f_1 = 0.75$, $c_1 = c_2 = 1$ and $\bar{d} = 2.7$ is shown in Table II. The corresponding ripple evolution is seen in Fig. 5 compared to the evolution proposed for the LT feedback code. There is still a slight lack of robustness near the feedback point. $Q(L)$ was found assuming that a ripple size equal to the desired is achieved. This is valid even when a slight deviation exists, as in Fig. 3. However, in this case the deviation has an impact and it can be compensated by transmitting the feedback slightly earlier. There is also a lack of robustness near the end of decoding. However, as in traditional Raptor codes, this is dealt with by an outer block code.

IV. Numerical Results

The performance metrics we are concerned with are average coding overhead, i.e. $(1+\epsilon)k$, and the average degree which determines the complexity. First, the performance of LT feedback codes is evaluated in the low range of k. This is due to the fact that traditional LT codes are inefficient for low k.

The first simulation serves the purpose of optimizing the feedback point for a single feedback opportunity. Fig. 6 shows the results over averages of 5000 runs for $c_1 = c_2 = 1$. The best performance was achieved with $f_1 = \frac{23}{32}$ for $k = 32$, 64 and 96. For higher k values, the best performance was achieved with $f_1 = \frac{24}{32}$. A similar optimization has been performed for two and three feedback opportunities. For two feedbacks, best performing values were $f_1 = \frac{18}{32}$ and $f_2 = \frac{28}{32}$ for $k = 32$, 64, 96 and 128. For $k = 160$, $f_1 = \frac{18}{32}$ and $f_2 = \frac{29}{32}$ performed best. For the case of three feedbacks, $f_1 = \frac{13}{32}$, $f_2 = \frac{24}{32}$, and $f_3 = \frac{30}{32}$ were either optimal or near optimal for all short k.

An average coding overhead is shown in Fig. 7. LT feedback codes with increasing number of feedbacks are compared to a traditional LT code with no feedback using the RSD. We can observe a significant gain of the feedback in average coding overhead. Even the LT feedback code with zero feedbacks provides a slight decrease in coding overhead due to the inefficiency of RSD. It is also evident that there is a diminishing return from adding feedback to the system.

$$\begin{bmatrix} q(1,k,k) & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & & & & \\ & & 0 & 0 & 0 & 0 \\ q(1,k'+1,k) & \cdots & q(f_1k,k'+1,k) & 0 & 0 & 0 \\ q(1,k',k) & \cdots & q(f_1k,k',k) & q(1,k',k') & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ q(1,1,k) & \cdots & q(f_1k,1,k) & q(1,1,k') & \cdots & q(k',1,k') \\ w\cdot 1 & \cdots & w\cdot f_1k & w\cdot 1 & \cdots & w\cdot k' \\ w\sum_{L=k'+1}^{k} q(1,L,k) & \cdots & wq(f_1k,k'+1,k) & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} n_1\Phi_1(1) \\ \vdots \\ n_1\Phi_1(f_1k) \\ n_2\Phi_2(1) \\ \vdots \\ n_2\Phi_2(k') \end{bmatrix} = \begin{bmatrix} Q(k) \\ \vdots \\ \\ \\ \\ Q(1) \\ w\bar{d} \\ w\sum_{L=k'+1}^{k} Q(L) \end{bmatrix}. \quad (11)$$

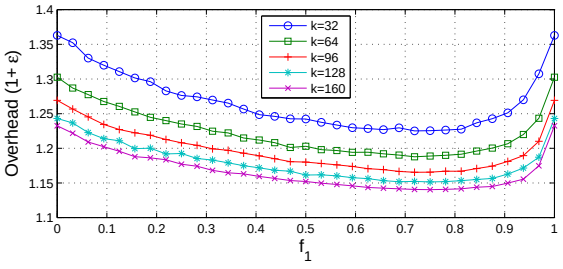

Fig. 6. Optimization of f_1 for the case of a single feedback opportunity.

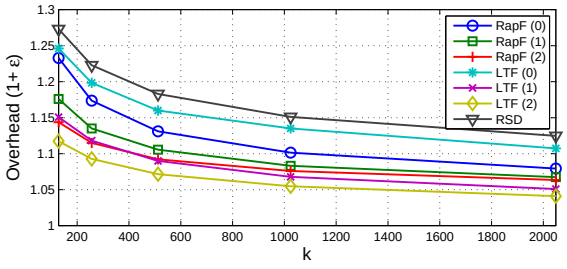

Fig. 8. Coding overhead of Raptor feedback codes and LT feedback codes.

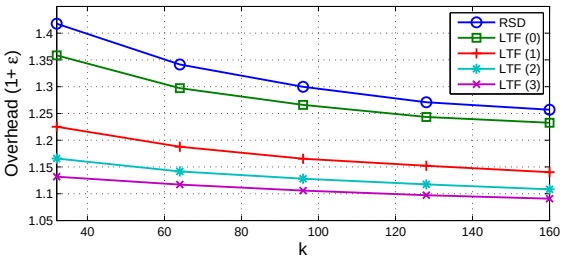

Fig. 7. Coding overhead of LT feedback codes and a traditional LT code.

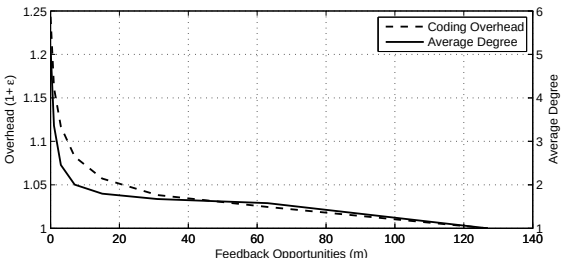

Fig. 9. Coding overhead and complexity of LT feedback code for increasing number of feedback opportunities.

In Fig. 8, Raptor feedback codes are compared to LT feedback codes at higher k values, where complexity becomes relevant to consider. For zero, one and two feedback opportunities in the Raptor feedback codes, $\bar{d}$ of 5, 3 and 2.5 have been chosen respectively. For all cases, a random LDPC code with variable nodes of degree 3 and a rate of 0.9688 has been applied as an outer code. Feedback points for all schemes have been optimized as described earlier. The results show that, except for the case of no feedback, the Raptor feedback code performs slightly worse than the LT feedback code. Hence, linear complexity comes at the price of an overhead loss, as is the case for traditional rateless codes.

Finally, Fig. 9 shows the average overhead and complexity of LT feedback codes for $k = 128$ as a function of the number of feedback opportunities. For this simulation, the feedback locations have not been optimized but are uniformly distributed over the decoding process. This figure shows that any amount of feedback opportunities can be utilized for decreasing overhead and complexity all the way towards the extreme case of ARQ, where $\bar{d} = 1$ and $\epsilon = 0$. Note that there is a room of improving the performance by optimizing the feedback points.

V. Conclusions

We have introduced LT feedback codes and Raptor feedback codes, those of which differ from traditional rateless codes through their ability to take any amount of feedback opportunities into account. The key component is the design of well-performing degree distributions for feedback-aided transmissions. We have shown significant improvements in coding overhead and complexity. Moreover, it was shown that LT feedback codes approach linear complexity for increasing number of feedback opportunities, which eliminates the major drawback of traditional LT codes. For a small number of feedback opportunities, complexity is still superlinear. For this case, Raptor feedback codes have been introduced in order to achieve linear complexity.

References

[1] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science.*, pp. 271–280, November 2002.

[2] A. Shokrollahi, "Raptor codes," *IEEE Trans. IT*, pp. 2551–2567, 2006.

[3] S. Kokalj-Filipović, P. Spasojević, E. Soljanin and R. Yates, "ARQ with doped fountain decoding," in *IEEE ISSSTA '08*, pp. 780–784, 2008.

[4] A. Beimel, S. Dolev and N. Singer, "RT oblivious erasure correcting," *IEEE/ACM Transactions on Networking.*, pp. 1321–1332, 2007.

[5] A. Hagedorn, S. Agarwal, D. Starobinski, and A. Trachtenberg, "Rateless coding with feedback," in *IEEE INFOCOM*, pp. 1791–1799, April 2009.