Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon Small Block FEC Codec

Vincent Roca, Christoph Neumann

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon Small Block FEC Codec

Vincent ROCA — Christoph NEUMANN
INRIA Rhône-Alpes, Planète Research Team, France
{firstname.name}@inrialpes.fr

http://www.inrialpes.fr/planete/

N° 5225

June 9, 2004

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5225--FR+ENG

Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon Small Block FEC Codec*

Vincent ROCA , Christoph NEUMANN
INRIA Rhône-Alpes, Planète Research Team, France
{firstname.name}@inrialpes.fr
http://www.inrialpes.fr/planete/

Thème 1 — Réseaux et systèmes
Projet Planete

Rapport de recherche n° 5225 — June 9, 2004 — 24 pages

Abstract: This paper introduces with a great level of details the design and the performance evaluation of four large block FEC codes: regular LDPC (Low Density Parity Check), LDGM (Low Density Generator Matrix), LDGM Staircase and LDGM Triangle, that are all capable of operating on source blocks that are composed of several tens of thousand of packets. If all four belong to the category of codes on graphs, they differ by their encoding algorithm. We evaluate and compare the performances of all four codes, as well as a Reed-Solomon codec which serves as a reference and gives an idea of the level of performance achievable with small block codes. The comparison focuses on three major aspects: the global decoding inefficiency (caused either by the FEC code or by the need to split the object into several blocks), the encoding and decoding speeds, and the memory requirements at the encoder and decoder. From these performance results, we identify the situations where they best operate.

Unlike many other works in the coding theory area, this paper deliberately skips theoretical aspects to focus on practical results. Besides this work does not consider techniques that are known to be patented (e.g. the use of irregular graphs), no matter how efficient they may be. This deliberate choice stems from our desire to design patent-free, open source FEC codecs. Our conclusion is that the LDGM Staircase/Triangle codes have excellent performances and open new opportunities in the area of bulk data distribution or storage.

Key-words: FEC, large block FEC codes, LDPC, LDGM, Reed Solomon

* This work is supported by a research contract with STMicroelectronics.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Conception, Evaluation et Comparaison de quatre codecs FEC de type grands blocs, LDPC, LDGM, LDGM Staircase et LDGM Triangle, ainsi que d'un Codec Reed-Solomon de type petits blocs[†]

Résumé : Ce papier introduit avec force détails la conception et l'évaluation de quatre codes FEC de type grand blocs: LDPC (Low Density Parity Check), LDGM (Low Density Generator Matrix), LDGM Escalier et LDGM Triangle, tous quatre capables d'opérer sur des blocs de données composés de plusieurs dizaines de milliers de paquets. Si ces quatre codes font partie de la catégorie des codes en graphes, ils diffèrent par leur algorithme d'encodage. Nous évaluons et comparons leurs performances, de même que celles d'un codec Reed-Solomon qui sert de référence et donne une idée du niveau de performances accessible avec des codes de type petits blocs. La comparaison porte sur trois aspects principaux: le ratio global d'inefficacité de décodage (dû soit au code FEC, soit à la nécessité de segmenter l'objet en plusieurs blocs), les vitesses d'encodage et de décodage, et les besoins mémoire coté encodeur et décodeur. De ces résultats nous identifions les situations où les codes sont les plus efficaces.

Contrairement à de nombreux autres travaux dans le domaine de la théorie du codage, ce papier évite délibérément les aspects théoriques pour se focaliser sur les résultats pratiques. En outres nous ne considèrons pas ici les techniques connues pour être brevetées (ainsi le fait de travailler sur des graphes irréguliers), quelle que soit leur efficacité supposée ou connue. Ce choix délibéré est issu de notre désir de proposer des codecs FEC "open source" libres de tout droit. Nos conclusions sont que les codes LDGM Escalier et Triangle ont tous deux d'excellentes performances et ouvrent de nombreuses opportunités dans le domaine de la transmission ou du stockage de gros contenus.

Mots-clés : FEC, codes FEC de type grands blocs, LDPC, LDGM, Reed Solomon

[†] Ce travail est supporté par un contrat de recherche avec la société STMicroelectronics.

1 Introduction

1.1 Use of FEC in Reliable Multicast

Providing reliable multicast in a best-effort network where packet losses are frequent, such as the Internet, requires a reliable multicast protocol, whose specific function is to compensate for these losses. Two approaches are used to provide reliability to multicast transport protocols: (1) Automatic Repeat Request (ARQ), where lost packets are retransmitted, and (2) Forward Error Correction (FEC), which transmits redundant data (called parity data) along with the original data. Thanks to this redundancy, up to a certain number of missing packets can be recovered at the receiver. More precisely, with a $(n;k)$ FEC code, k source packets are encoded into n packets, and the additional $n-k$ packets are called parity packets. A receiver can then build the k source packets provided it receives any k packets out of n (or a little bit more than k with LDPC/LDGM codes). The great advantage of using FEC is that the same parity packet can be used to recover different lost packets at different receivers. Therefore, feedback from receivers can be implemented at a coarser granularity, which in particular limits the risk of feedback implosion at the source. An extreme case is the *Asynchronous Layered Coding* (ALC) massively scalable reliable multicast protocol [11, 12] which achieves reliability thanks to an intensive use of FEC. There is absolutely no feedback information from the receivers in that case.

The first motivation for this work is our *MultiCast Library (MCLv3)* [25] which implements the ALC protocol family and provides an integrated solution for the reliable and highly scalable multicast delivery of bulk data. Previous work from Digital Fountain [2] or the RMT IETF working group [15], and our own experience gained with MCLv3 [28, 27], have highlighted the importance of FEC and the two limitations of the Reed-Solomon FEC code (or RSE) commonly used: (1) the small block size and (2) the high encoding/decoding times. The two parameters $(k;n)$ play an important role in the global efficiency accessible:

- a large n is favorable because it reduces the probability of packet duplication at a receiver. Indeed, since the same set of packets (data or parity) may be transmitted in a different random order in each ALC layer [28][1], the probability that a receiver receives the same packet on two different layers decreases if n increases. The extreme case is achieved with the so-called rate-less codes that can produce an infinite number of parity packets (i.e. $n \rightarrow +\infty$). If these codes are of not considered in our performance evaluation, we consider them in our discussion (section 4.3).

- a large k increases the correction capacity of the FEC code. Since a parity packet can only recover an erasure in the block it belongs to, its erasure capability is therefore inversely proportional to the number of blocks a file must be segmented into. Ideally a file may be encoded in a single block, in which case each parity packet would be useful.

Unfortunately RSE is intrinsically limited by the Galois Field size it uses. A typical example is GF(8) where $n \leq 256$. With one kilobyte packets, a FEC codec producing as many parity packets as data packets ($n = 2k$) operates on blocks of size 128 kilobytes at most, and all files exceeding this threshold must be segmented into several blocks. Using GF(16) solves this problem since here $n \leq 65536$. But a major drawback then is a huge encoding/decoding time. Therefore the k and n parameters must be limited to small values if high transmission rates are desired.

Yet RSE achieves an ideal encoding because the portion of the transmission sufficient to recover the original message is the length of the original message itself (said differently, k packets are always sufficient for decoding to complete). A FEC code with this property is called MDS, or Minimum Distance Separation.

1.2 Use of FEC in Environments with Intermittent Connectivity

Several environments are characterized by an intermittent connectivity. [3] provides an example which nicely fits in this category: the Land Mobile Satellite (LMS) channel. The goal of the application is

[1] Note that other possibilities exist to bypass the limitations of RSE codes [1] or with new expandable codes (also known as rate-less codes) [18, 10] that will be discussed later on.

to distribute multimedia contents to the car passengers from a satellite connection. In this system, the mobile (car) experiences both fast fading and white Gaussian noise, which is addressed by the DVB-S error correction schemes, and slow fading (e.g. when entering a tunnel), which is addressed by a packet level FEC encoding. This two level FEC encoding is fully complementary, each level of protection addressing different problems. As mentioned in [3], the LDGM codes that we discuss in our paper are well suited to this kind of environment.

1.3 Use of FEC in Distributed Storage

Another area where FEC proves to be of great help is distributed storage systems, where the object is replicated across a set of servers. Here clients download the object by accessing multiple servers in parallel. Such replicated systems improve both fault-tolerance and performance over non replicated storage systems. Using an FEC erasure code to dilute the information over the storage servers allows a greater and flexible choice to the clients, and can enhance the overall downloading throughput.

Basically, for each of the B blocks, FEC encoding produces $n = fec\_ratio * k$ packets, where the FEC expansion ratio is an integer ≥ 1. The system can now replicate any of the $B * fec\_ratio$ "encoding blocks" on the servers, and downloading any B different "encoding blocks" out of $B * fec\_ratio$ possibilities is sufficient to reconstruct the original file.

Several such systems, relying on various kinds of FEC codes have been proposed, like [7, 8, 19, 23] to cite only a few.

1.4 LDPC codes for Packet Erasure Channels: a Bit of History

LDPC codes were first introduced by Gallager in 1960 [4, 5]. However LDPC has been almost completely forgotten for the next 30's years and has only been rediscovered in 1995 by MacKay and Neal [17]. LDPC has then been largely improved by Luby, Shokrollahi and al. This work lead to the design of such codes as Tornado[2], LT[10], and Raptor [29]. Yet all three codes are protected by patents (see [15] for a non exhaustive list of patents). Because of these limitations, we do not consider them in the present work but focus on LDPC and several derivatives.

LDPC can operate both over Binary Symmetric Channels (BSC) and Binary Erasure Channels (BEC). With a BSC channel a bit arrives at the destination either perfectly or erroneously. It typically happens over noisy channels. On the opposite, in a BEC channel a bit either arrives perfectly or is lost by the channel. A typical example of erasure channel, which does not operate on bit streams but on packet streams, is the Internet. In that case the CRC available in most physical layers guaranties that a packet arriving at its destination and given to the upper layer is error-free, otherwise it would have been destroyed. The main cause of packet losses, in that case, is router congestion.

LDPC, like many other FEC codes, can operate on both types of channels. Yet LDPC is largely simplified when we only consider the case of packet (or bit) erasure channels (resp. PEC or BEC), since in that case decoding can exploit the additional information that does not exist over symmetric channels: a packet (or bit) received is error-free. In the remaining of this paper we only consider the PEC case.

Note that the term "symbol" is often used in coding theory. Symbols refer to bits when operating over a binary channel (BEC or BSC), and packets when operating over a packet erasure channel (PEC). Since we only consider the PEC case in this work, we will use the term packet rather than symbol, but most of our results are also valid over a BEC.

1.5 Goal of this Work and Choices Made, in Particular Concerning Patented Techniques

In this paper we describe the implementation details and the performances of several large block FEC codes derived from LDPC. These codes have two main advantages: (1) they use XOR operations for high speed encoding/decoding, and (2) they operate on large source blocks ($k \gg 1$).

In the present work we deliberately *do not consider techniques which are known to be patented, no matter how efficient they may be.* This is the case of the use of irregular graphs [13] in LDPC derived

FEC codes, which is covered in particular by [14]. This choice stems from our desire to *design patent-free, open source FEC codecs*. Besides we want to show that the performance level of these patent free codes are already good enough for them to be used in several environments (and we remain rather confident on the possibility to further improve them).

Unlike many other works in the coding theory area, this paper deliberately skips theoretical aspects to focus on practical results. This standpoint is only rarely considered and [3, 23], our preliminary work [26], and to some extent [2], are the only similar works we are aware of.

In this work we will focus on:

- the global decoding inefficiency, caused either by the LDPC/LDGM FEC code or by the need to split the object into several blocks with RSE;
- the encoding and decoding speeds, which also determines the CPU load required by these codes, and hence their ability to be used either on high performance links or, at the contrary, on lightweight hosts;
- and the memory requirements at both the encoder and decoder.

From these performance results, we identify the situations where each code operates the best, if any, and discuss some of their fundamental requirements and limits.

A direct effect of patents on FEC codes is that, to the best of our knowledge, no public implementation of any Digital Fountain's codes exist, that could have been used for our performance evaluations. We therefore only consider the raw figures provided in [2, 23] without any possibility to explore some of the performance metrics or operational conditions that we believe are important but lack in these works.

The remainder of the paper is organized as follows: We first introduce the four large block codes considered in section 2. We compare their performances and that of a traditional Reed-Solomon codec in section 3. We discuss our results and that of related works in section 4. Finally we conclude.

2 The LDGM, LDGM Staircase, LDGM Triangle and LDPC FEC Codes

In this section we introduce successively four codes on graphs: LDGM, two variants, LDGM Staircase and LDGM Triangle, and finally LDPC, the general case.

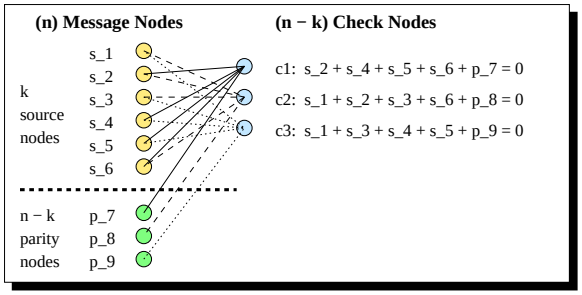
2.1 LDGM Code

The Low Density Generator Matrix (LDGM) code [26] is a linear $(n; k)$ block code. It creates linear equations between the k source packets: s_i, $i \in \{0..k-1\}$ and the $n-k$ parity packets: p_i, $i \in \{k..n-1\}$. More precisely each parity packet is equal to the sum of a subset of the source packets, as defined in the linear equations. The relationships specified by these equations can be represented using one of two equivalent representations: a *bipartite graph* and a *parity check matrix* (figure 1).

2.1.1 Bipartite Graph

The bipartite graph defines relationships between left nodes, called message nodes, and right nodes, called check nodes. The k source packets constitute the first k message nodes, while the parity packets constitute the remaining $n-k$ message nodes. The check nodes represent relationships between the various packets (but are not themselves packets) and form a set of linear equations. For each edge in this graph between a source message node and a check node, the corresponding source node is sum'ed (XOR'ed) in the associated constraint. In the LDGM approach, all the parity message nodes are linked to *exactly one* check node (it is not the case with other codes). Since we only consider regular codes, the degree of each source message node, also called *left degree*, is the same, and the degree of check nodes is on average equal to:

$$right\_deg_{LDGM} = 1 + \frac{k \times left\_deg}{n-k}$$



(a) Bipartite graph

$$[\,H_1 \mid \mathrm{Id}_3\,] = \left(\begin{array}{cccccc|ccc} s_1 & & & & s_6 & & p_7 & \cdots & p_9 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right) \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \end{array}$$

(b) Dual Parity Check Matrix

Figure 1: Example of regular bipartite graph and the associated parity check matrix for LDGM.

When using a high FEC expansion ratio (i.e. a large $n-k$ denominator in the above equation), there may be check nodes (i.e. equations) connecting less than two source packets. In such a case, we automatically add one or two extra edges in the bipartite graph with randomly selected source packets. Therefore the average right degree is at least 2 in the above equation ($right\_deg_{LDGM} = max(2; ...)$).

We do not try to remove degree four cycles in the parity check creation process. The main motivations for this choice are (1) we do not want to compromise encoding and decoding performances (this is a CPU intensive task), and (2) we observed little influence over the overall performances, which is confirmed by [3].

These two rules also apply to the other LDGM or LDPC codes introduced in the coming sections.

2.1.2 Parity Check Matrix

A dual representation consists in building a parity check matrix, H. With LDGM, this matrix is the concatenation of matrix H_1 and an identity matrix I_{n-k}. There is a 1 in the $\{i; j\}$ entry of matrix H each time there is an edge between message node j and check node i in the associated bipartite graph.

Thanks to the `XOR` properties, and thanks to the I_{n-k} identity matrix, encoding (i.e. parity packet creation) is straightforward: each parity packet is equal to the sum of all source packets in the associated equation. For instance, packet p_7 is equal to the sum: $s_2 \oplus s_4 \oplus s_5 \oplus s_6$.

The fact that this matrix can be used for encoding, or in other words the fact this matrix is a "Generator Matrix", accounts for the "GM" acronym in the LDGM name. Besides, the H matrix is sparse[2]. This is due to the fixed number of 1s per column (at most 10 in all simulations presented in this paper), no matter the matrix size, or said differently, no matter the block size (k) and number of parity packets created ($n-k$). This "low density" property accounts for the "LD" acronym in the LDGM name. Because of these two properties, *encoding is extremely fast with LDGM and its derivatives.*

2.1.3 Decoding with the Iterative Decoding Algorithm

Decoding follows a trivial algorithm. Given a set of linear equations, if one of them has only one remaining unknown variable, then the value of this variable is that of the constant term. We then replace this variable by its value in all remaining equations and reiterate, recursively. The value of several variables can therefore be found if lucky.

In our case, the set of constraints (check nodes) form a set of k linear equations of n variables (source and parity packets). As such this system cannot be solved and we need to receive packets from the network. Each incoming packet contains the value of the associated variable, so we replace this variable in all linear equations in which it appears. We then apply the above algorithm and see if decoding can progress by one or more steps. As we approach the end of decoding, incoming packets tend to trigger the

[2]Note that this sparse property is not visible in figure 1(b) which has a limited size for practical reasons.

decoding of several packets, until all of the k source packets have been recovered (but not necessarily all of the $n-k$ parity packets).

2.2 LDGM Staircase Code

LDGM Staircase is a trivial variant of LDGM which is suggested in [16][3]. LDGM Staircase only differs from LDGM by the fact that the I_{n-k} matrix has been replaced by a "staircase matrix" of the same size (figure 2 (a)).

This small variation does not affect encoding, which remains a simple and highly efficient process. The only constraint is that encoding must follow the natural parity packet order. For instance encoding starts with packet p_7, and then packet p_8 (which can be computed since p_7 is known), etc. Yet, as will be shown later, this simple variation *largely improves the decoding efficiency*. Intuitively, parity packets are now protected and an erased parity packet, for instance packet p_7, can be recovered thanks to packet p_8 since they are both linked by a common constraint (node c_2). On the opposite, with LDGM, an erased parity packet cannot be recovered, unless all the associated source nodes in the associated constraint are known, but in that case this is useless! Decoding follows the same algorithm as LDGM.

Given the right degree, the left degree of check nodes is on average equal to:

$$right\_deg_{LDGM\_Staircase} = 2 + \frac{k \times left\_deg}{n-k}$$

2.3 LDGM Triangle Code

LDGM Triangle is a variant of LDGM Staircase. The empty Triangle located beneath the Staircase diagonal in the LDGM Staircase case is now filled, following an appropriate rule given below:

```
/* For row i, assuming that matrix H1 is already filled. */
add a 1 in row i and column k+i, and if i>0 in column k+i-1; // to create the staircase
tmp_idx = i-2;
for (n = i-2; n > 0; n--) {                                  // add at most i-2 "1"s
    tmp_idx = rand() % tmp_idx;
    if (tmp_idx > 0)
        add a 1 in row i and column (k + tmp_idx);           // to create the triangle
    else
        break;
}
```

This rule leads parity nodes with a low index number to have a higher degree than parity nodes with a higher index number, as shown in figure 2. This variation increases performances compared to LDGM Staircase for small FEC expansion ratios as will be shown in section 3.4. Intuitively, we add a "progressive" dependency between check nodes, so that only some check nodes protect a larger number of nodes[4].

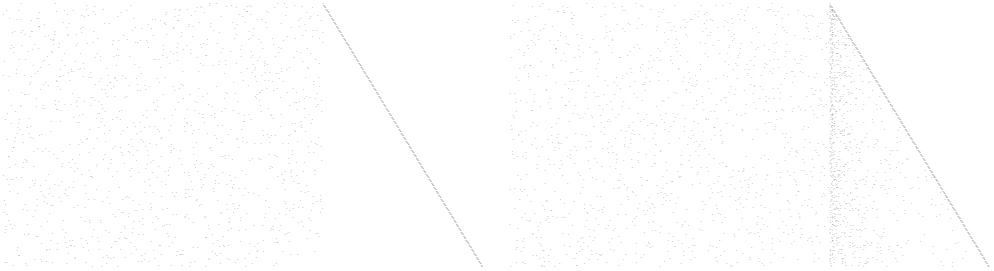
Here also encoding must follow the natural parity packet order (this is the reason why only the lower triangle is filled). Otherwise it remains highly efficient, even if a bit slower than with LDGM Staircase since there are more "1"s per row. Decoding follows the same algorithm as LDGM.

2.4 LDPC Code

LDPC differ from the three previous codes by the fact the H_1 matrix encompasses *all* message nodes, source and parity packets (so H and H_1 are the same). A parity packet now takes part in several constraints randomly selected.

[3]The author of [16] refers this code as LDPC Staircase, we prefer the LDGM Staircase name because of its closeness to LDGM and the fact that the parity check matrix H is also a generator matrix

[4]It is interesting to note that experiments (not included in this paper) have shown that filling the triangle in a completely random manner, without progression in the density, lead to lower performances than LDGM Staircase.



(a) LDGM Staircase (b) LDGM Triangle

Figure 2: Example of parity check matrix (H) for LDGM Staircase and Triangle (k=400, n=600).

This is an asset from a decoding point of view since an erased parity packet may be recovered by several different ways (as with LDGM Staircase/Triangle). But it has a cost since there is no trivial encoding scheme and the parity check matrix cannot be used directly for producing parity packets. Encoding requires that the set of $n-k$ linear equations be solved, where the $n-k$ parity packets take the role of variables. Several schemes exist to that purpose [16] that all produce a generator matrix, G, solution of the system, but even with optimized schemes, it remains a time consuming task. Encoding now consists in multiplying this G matrix with the vector composed of source packets. Since G has no reason to be sparse (unlike H), this multiplication is yet another time consuming task. Non surprisingly, *encoding is rather slow with LDPC compared to LDGM codes.*

Decoding follows the same algorithm as LDGM (section 2.1.3) and operates on the H matrix.

Because of our choice of not using patented irregular graphs, we only consider in this work regular (3, $right\_deg\_LDPC$) LDPC codes, where the left nodes of the bipartite graph are of degree 3 and right nodes of degree:

$$right\_deg\_LDPC = \frac{n \times left\_deg}{n-k}$$

3 Performance Evaluation of the Four LDPC/LDGM FEC Codecs plus an RSE codec

3.1 Codec Design

We have designed an open source C++ LDGM/LDGM Staircase/LDGM Triangle/LDPC codec [21]. This codec relies on R. Neal's software [20], which, given the $k, n, left\_degree$ and a *seed*, creates a regular parity check matrix H. This matrix is then used by the coder and decoder parts of our codec, either directly (LDGM/LDGM Staircase/LDGM Triangle), or through the G generator matrix (LDPC at a sender). Our codec is implemented as a library that has to be included by another software, either an application (see the demo applications included in the distribution [21]), or a reliable multicast implementation (see our MCLv3 library implementing the FLUTE/ALC family of protocols [25]).

So far our codec assumes that all source and parity packets are stored in physical memory (RAM), which creates a practical limitation for huge objects (section 3.6). Future work will address this aspect and will use the dedicated virtual memory framework included in MCLv3 [25]. There is anyway a limitation because the random nature of the H/G matrices requires at any time a fast access to any source packet (and for LDPC and LDGM-Triangle codes also to any parity packet) during the encoding and decoding processes, which is only possible if most of the packets are kept in memory.

The matrix storage scheme is critical for achieving good performances. We used for that the sparse/dense library included in R. Neal's software. The H parity check matrix is stored in a sparse array: to each

"1" of the matrix is associated an entry which contains its row and column indexes, plus a pointer to the following "1" on the left, on the right, upward[5] and downward. This sparse structure has two advantages:

- it enables major memory savings compared to a dense structure, (e.g. a $k = 20000; n = 30000$ code leads to a H matrix of size 10000×30000 which requires at least 37.5 MBytes of storage with a dense representation);
- it also enables a very fast encoding (since it is sufficient to follow the "right" pointers of a given line) and decoding (since it is sufficient to follow the "down" pointer of a given column, i.e. packet, to find all the equations where it takes part).

With LDPC, the sparse structure is not appropriate for storing the G generator matrix (which has no reason to be sparse), and in that case a dense structure is used. This has a cost which is in parts responsible of the limitations observed during tests.

3.2 Parameters and Performance Metrics Considered

The following parameters are considered during tests:

- *FEC expansion ratio, fec_ratio, and code rate:* the fec_ratio is the n/k ratio. The higher the ratio, the more parity packets are created. This FEC expansion ratio is also called "stretch factor" in [2]. This ratio is the inverse of the "code rate", defined as k/n and which is widely used in the coding community. FEC codes capable of producing an infinite number of parity packets have a rate equal to zero and are therefore called "rate-less". In the present work we will use the fec_ratio terminology.
- *object size:* in this work packets are always 1024 bytes long. We therefore express the object size in packets rather than in bytes. With LDPC/LDGM codes, the object being encoded as a single block, the object size in packets is equivalent to the k parameter of the FEC code. With RSE, the object is split into several blocks, each of size k packets.

We measured the following performance metrics:

- *global decoding inefficiency ratio, $inef\_ratio$:* decoding inefficiency can be caused by:
  - the fact the FEC code is non MDS (e.g. LDPC/LDGM codes), so more than k packets must be received for the decoding a block of size k to complete (this is also known as the "overhead factor" in [23]), or
  - the need to split big objects into smaller blocks (e.g. RSE).

  In both cases $inef\_ratio * number\ of\ source\ packets$ packets, with $inef\_ratio \geq 1$, must be received in order to decode (all) the block(s). Said differently:

$$inef\_ratio = \frac{number\ of\ pkts\ required\ for\ decoding}{number\ of\ source\ pkts}$$

  This $inef\_ratio$, experimentally evaluated, is one of the key performance metrics we consider.
- *decoding inefficiency ratio confidence intervals:* considering the average $inef\_ratio$ only is not sufficient since large values are sometimes observed. We therefore measure various confidence intervals for values larger than the average, and the same for values smaller than the average.
- *encoding/decoding times and bandwidths:* measuring the FEC codec initialization time and the total encoding (resp. decoding) time enables to measure the maximum sustainable bandwidth. More precisely, two encoding bandwidths can be considered: a global encoding bandwidth measured as:

$$global\_encoding\_bw = \frac{n * pkt\ size\ (in\ bits)}{time}$$

[5] Except with LDGM/LDGM Staircase codes where the upward pointer is removed.

and a partial encoding bandwidth which only takes into account the number of parity packets produced:

$$partial\_encoding\_bw = \frac{(n-k) * pkt\ size\ (in\ bits)}{time}$$

The first measure indicates whether the codec can support a given transmission rate when using a systematic FEC code (the default, both source and parity packets are sent), whereas the second one refers to what is actually *produced* by the FEC codec.

No such distinction exists at the receiver, and the bandwidth is always measured as:

$$\frac{number\ of\ pkts\ required\ for\ decoding * pkt\ size\ (in\ bits)}{time}$$

Since this value depends on the global decoding inefficiency ratio which varies slightly at each test, we only consider average values.

- *maximum memory requirements:* we measure the maximum memory requirements, both during encoding and decoding, caused by the need to store source packets (and often parity packets), but also all the codec's internal buffers or tables, and in particular the parity check matrix with LDGM/LDPC codes.

3.3 Experimental Setup

3.3.1 Testing Application

The following tests use both our LDPC/LDGM codec and a widely used RSE codec [24]. We designed a basic application on top of these two codecs, derived from the `perf_tool` of our FEC codec distribution [21]. Given an object and a fec_ratio, the application first performs FEC encoding: to k source packets of a block, the codec produces $n-k = (fec\_ratio-1)*k$ parity packets. Both source and parity packets are 1024 byte long. Once encoding has been performed, the source and parity packets (of all blocks with RSE) are transmitted in a fully random order. The receiving application waits until it has received enough packets to decode the block (or all blocks with RSE). He then stops reception and reports the number of packets received and the global $inef\_ratio$ measured.

Several remarks can be made regarding this methodology:

- There is no explicit loss simulation model (e.g. random or per burst) because random transmissions are not affected by the loss model. Performance results could be slightly different if other transmission models were used, for instance by sending all k packets in sequence except a few of them (assumed erased) and then parity packets. But this methodology has a major drawback: it introduces additional parameters like the loss model and the loss ratio(s). On the contrary our approach avoids these problems, is more universal, and is in line with the way these codes can be used for instance in ALC sessions [6, 28].
- The same application and environment is used for testing all codes, including RSE.
- No transmissions on the network are performed, and the same application performs both encoding and decoding. This feature enables us to better control the way packets are "sent", simplifies tests, reduces the overall test durations and enables accurate encoding/decoding time measurements.

3.3.2 Determining Each Block Size with RSE

Because of the intrinsic limitations of a RSE codec working on $GF(8)$ [6], a large object has to be fragmented into several blocks compatible with the codec limitations: $k \leq n \leq max\_n = 256$. Determining the block structure of an object requires two things. The first key parameter is the maximum source block size possible, max_k. Two strategies exist for that:

[6] Using $GF(16)$ enables to use larger blocks, since $n \leq 65536$, but this is at the price of higher encoding/decoding times, whereas these times are already a major limitation with $GF(8)$ (section 3.5).

- for a given FEC expansion ratio, we compute: $max\_k = max\_n/fec\_ratio = 256/fec\_ratio$. Doing so leads to an optimal use of the RSE codec, since we are sure to always operate with blocks of maximum size.
- or we set max_k to a fixed value and produce the required $n - max\_k$ parity packets, as long as $n \leq max\_n$ (e.g. we can have a FEC ratio of at most 5 if $max\_k = 51$). This scheme does not enable an optimal use of the RSE codec, since larger blocks could often be possible, but a fixed, smaller max_k is sometimes chosen because of performance issues (see table 1).

The second key feature is the blocking algorithm. We chose the one specified in the FLUTE standard [22] which is known to be optimal since this scheme avoids situations where the final block is much smaller than all previous maximum size blocks, a situation largely sub-optimal (an extreme case is a last block of size one packet only). Here all blocks are of the same size, $A\_large \leq max\_k$, or one packet less than A_large.

Taking care of these two aspects is of high importance for reliably comparing the performances of the RSE and LDPC/LDGM codes, and we consider both ways of determining max_k in our tests.

3.3.3 Tests Carried Out

We conducted two kinds of tests:

- *Fixed FEC expansion ratio and different object sizes:* the fec_ratio is set to 1.5, while the object size varies from 100 packets (100 KBytes) up to 50000 packets (≈50 MBytes) [7].
- *Fixed object size and different FEC expansion ratios:* the object size is set to 20000 packets (≈20 MBytes), while the fec_ratio varies from 1.25 up to 5.0 (or, equivalently, the code rate varies from 0.8 down to 0.2).

During these experiments we measure the minimum number of packets required for decoding, and the encoding/decoding times. Experiments are repeated 1000 times with LDGM, LDGM Staircase, LDGM Triangle and RSE. We limit the number of LDPC experiments to 100 only due to performance issues (see section 3.5). The average value, and the 99% confidence interval for the values lower or higher than this average, are reported. We also discuss the absolute minimum/maximum values and higher confidence intervals (99.9% and 99.99%) in section 3.4.3.

Note that our transmission scheme leads to a *maximum decoding inefficiency equal to* $n/k = fec\_ratio$ since the number of packets required for decoding is at most equal to the number of packets transmitted, $n = fec\_ratio * k$.

3.4 Decoding Inefficiency Experiments

3.4.1 Fixed FEC ratio and different object sizes

We first compare the results using a fixed FEC ratio. Results are given in figure 3. We see that LDGM Triangle obtains the best results, with an inefficiency ratio that quickly stabilizes around 1.055. The 99% confidence interval is quite small, $[1.051; 1.059]$, with an object of size $50,000$ packets.

Like LDGM Triangle, the inefficiency ratios of LDGM Staircase and LDPC decrease and quickly stabilize when the object size increases. At the same time the confidence interval gets smaller. LDGM Staircase is a bit less efficient than its triangle variant, with an inefficiency ratio around 1.068 and a 99% confidence interval of $[1.064; 1.071]$ with an object of size $50,000$ packets. LDPC's inefficiency ratio is a bit higher, around 1.076, with a 99% confidence interval of $[1.073; 1.079]$.

LDGM behaves differently than the other three codes: its inefficiency ratio is much worse and increases slowly with the object size (a result already observed in our previous work [26]), the confidence intervals are larger, and the overall performances depend on the left degree parameter. 7 is the optimal degree in this case [26], and the LDGM inefficiency ratio is around 1.155 with an object of size $50,000$ packets.

[7] Note that unlike [23] we do not consider small objects, since small block FEC codes like RSE are in that case preferable to large block codes.

But changing the FEC expansion ratio also changes this optimal left degree. No such dependency exists with the other large block FEC codes where a left degree of 3 always yields the best results, which is in line with the results mentioned in the literature for LDPC.

Finally figure 3 (e) and (f) shows that RSE performances depend on the object size: the larger the object, the lower the efficiency, which is not a surprise (section 1.1). This has to be compared with the constant decoding inefficiency observed with LDGM Staircase/Triangle codecs[8]. With an object of size $50,000$ packets, RSE has a decoding inefficiency around 1.122 with fixed n, and 1.220 with fixed k, which is far behind LDGM Triangle. Besides the confidence intervals are much higher with RSE. On the opposite, with very small objects, RSE remains the best choice, essentially because it's an MDS code.

3.4.2 Fixed object size and different FEC ratios

The behavior of the four large block FEC codes is much more differentiated if we vary the FEC expansion ratio. This is visible in figure 4. First we notice that the LDPC curve stops at a FEC ratio of 3.0. This is due to a pure practical reason and shows a major limitation of the LDPC codec used: LDPC has a very high memory consumption, and above a FEC ratio of 3.0 the required memory exceeds the 2GB RAM of our machine. Moreover we see that even for FEC ratios smaller than 3.0, LDPC performances are not quite interesting, even if the confidence interval is small.

LDGM with a left degree equal to 10 has also poor performances, with an inefficiency ratio of 3.045 when the FEC expansion ratio is 5.0.

The LDGM Staircase and Triangle codes yield the most interesting results. *LDGM Triangle is the code that performs the best for FEC ratios smaller than 2.5.* At 2.5, LDGM Triangle has an average inefficiency ratio of 1.115 and a confidence interval going from 1.103 to 1.127. LDGM Staircase has an inefficiency ratio of 1.148 and a 99% confidence interval of $[1.138; 1.159]$ for the same FEC ratio.

For FEC ratios greater than 2.5, the performance of LDGM Triangle decreases suddenly. *On the opposite LDGM Staircase performs well for FEC ratios larger than 2.5.* LDGM Staircase reaches a local minimum at a FEC ratio of 2.75 (inefficiency ratio of 1.138) and behaves very well up to a FEC ratio of 5.0 (inefficiency ratio of 1.325, compared to 1.849 with LDGM Triangle), with a small confidence interval, $[1.314; 1.338]$.

Finally this kind of tests is more favorable to RSE than the previous ones with a fixed FEC expansion ratio, essentially because RSE is an MDS code (indeed, if there is a single block, producing more parity packets does not lead to performance degradation with an MDS code).

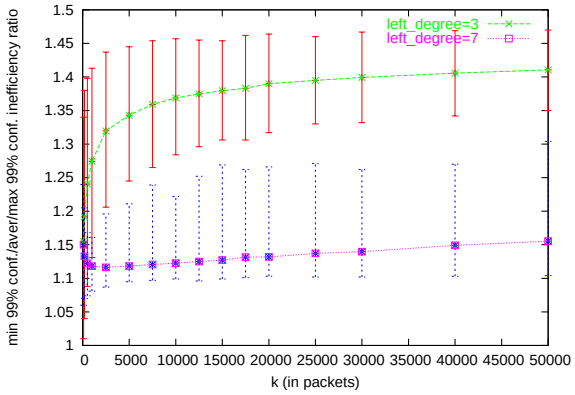
Yet, with the fixed n strategy (figure 4 (e)), the block size decreases if the FEC ratio increases (let's recall that $max\_k = 256/fec\_ratio$), there are more and more blocks and the global inefficiency ratio also increases. The inefficiency ratio is 1.241 with a FEC ratio of 2.5, and 1.512 with a FEC ratio of 5.0, which is worse than LDGM Staircase in both cases, and higher than LDGM Triangle in the first case. With a fixed k (figure 4 (f)) we observe that the global inefficiency ratio is worse with small FEC ratios, but about the same than with the fixed n strategy for higher FEC ratios (this is normal since the max_k value is about the same if $fec\_ratio = 5.0$). We also see that the confidence interval is rather large with RSE and increases with the FEC ratio, whereas no such behavior exists with other codes (except LDGM).

To conclude, RSE is not the best choice which remains LDGM Staircase for high FEC ratio values, and LDGM Triangle for small FEC ratio values.

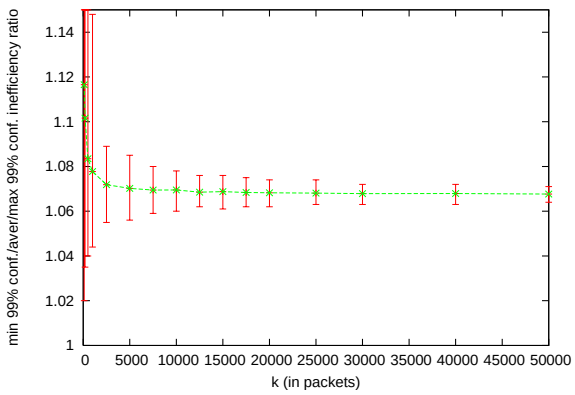
3.4.3 Variations in the Results

A result not highlighted on figures 3 and 4, is that the absolute maximum values (worst global inefficiency ratio) can largely exceed the average, even if this is extremely rare (less than 1%, since we generated these graphs with a confidence interval of 99%). These peaks appear with all three LDGM codecs, but their number and importance decreases with the block size. We did not notice this behavior with LDPC though, but this can be due to the fact we only did 100 tests, not 1000. No such behavior happen for

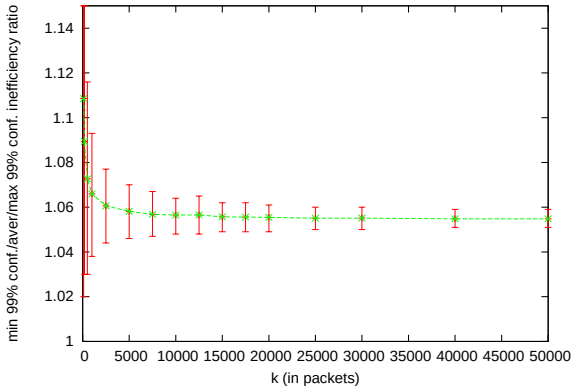
[8] At least as long as the maximum block size of such codes is not achieved, which essentially depends on the available memory size. Above this threshold, several blocks must be used, each of them being independently encoded.



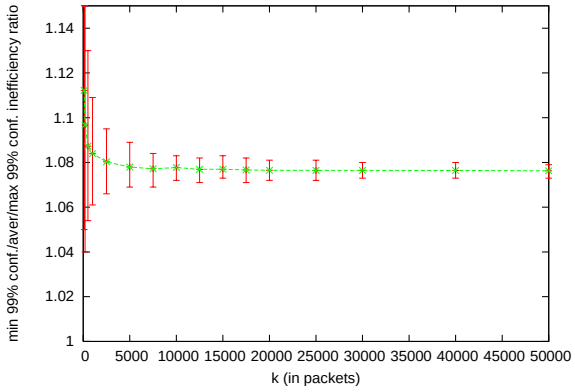
(a) Inefficiency ratio of LDGM



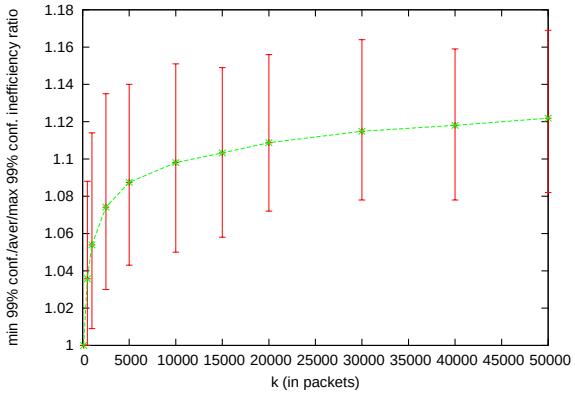
(b) Inefficiency ratio of LDGM Staircase



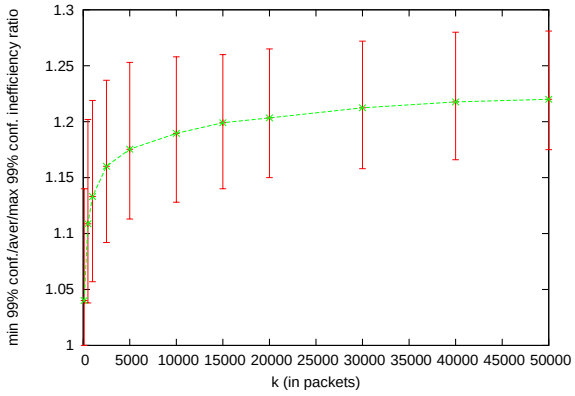
(c) Inefficiency ratio of LDGM Triangle



(d) Inefficiency ratio of LDPC

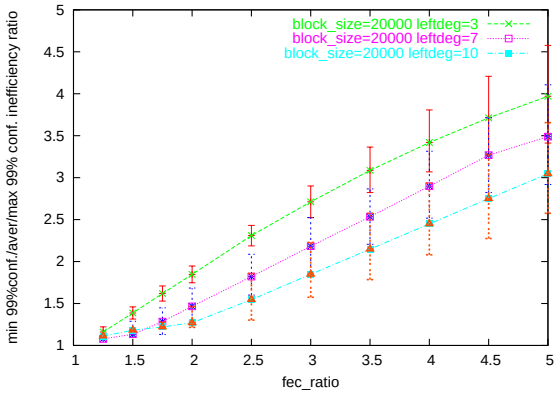


(e) Inefficiency ratio of RSE with fixed n(=256).

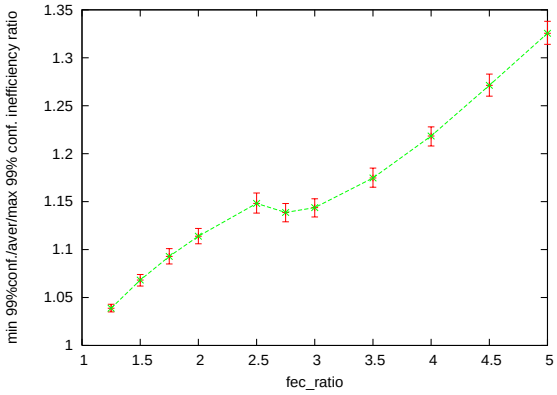


(f) Inefficiency ratio of RSE with fixed k(=51).

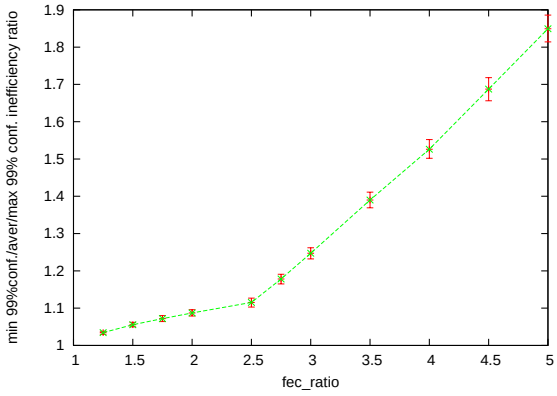
Figure 3: Global inefficiency ratio of LDPC/LDGM/RSE codes as a function of the object size, with FEC ratio = 1.5.



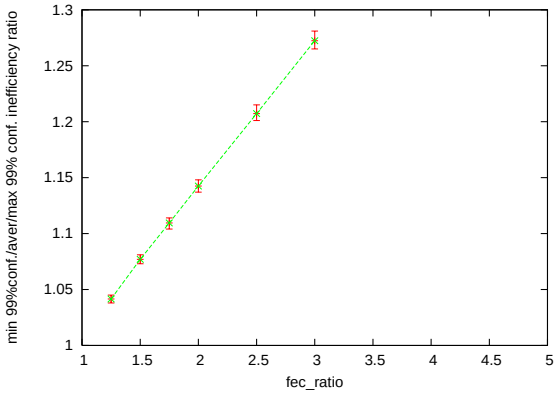
(a) Inefficiency ratio of LDGM



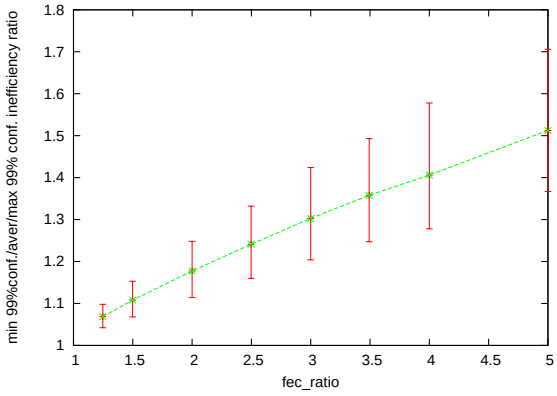
(b) Inefficiency ratio of LDGM Staircase



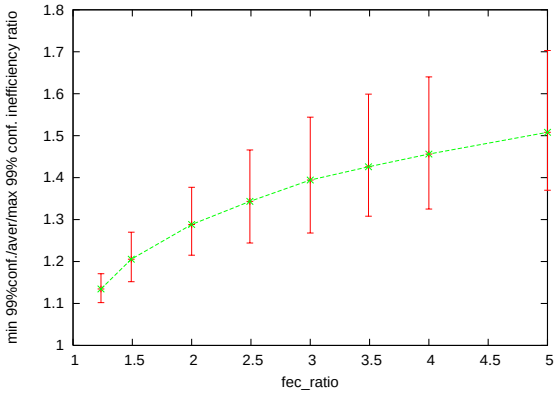
(c) Inefficiency ratio of LDGM Triangle



(d) Inefficiency ratio of LDPC



(e) Inefficiency ratio of RSE with fixed n(=256).



(f) Inefficiency ratio of RSE with fixed k(=51).

Figure 4: Global inefficiency ratio of LDPC/LDGM/RSE codes as a function of the FEC ratio, with object size = 20000 packets.

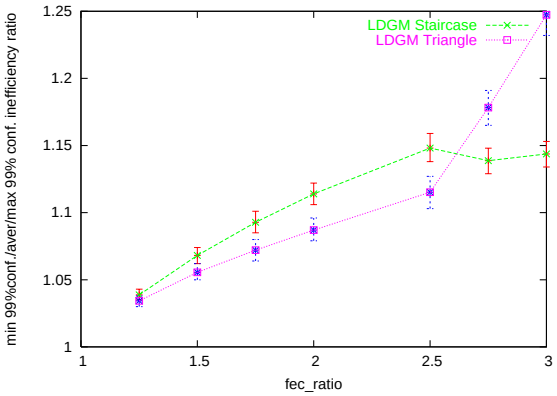
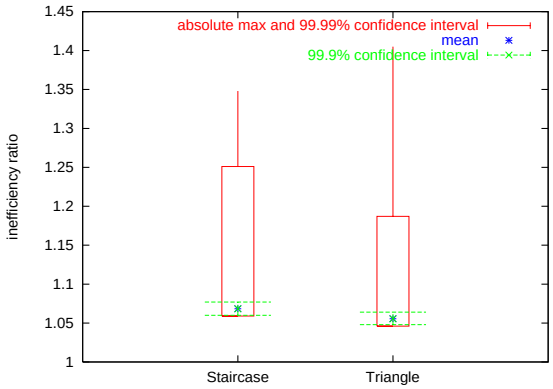
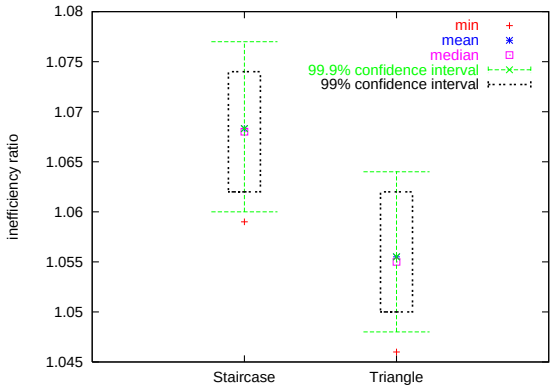


Figure 5: Global inefficiency ratio of LDGM Staircase and LDGM Triangle codes as a function of the FEC ratio (closeup).

the absolute minimum values (best inefficiency ratio), unfortunately. In order to quantify this behavior more precisely, we carried out other experiments with the LDGM Staircase and Triangle codes, with a fixed object size of $20,000$ packets and a FEC expansion ratio of 1.5, repeating the tests $100,000$ times. Results are shown in figure 6.



(a) average/max values and 99.99%/99.9% confidence intervals



(b) average/median/min values and 99.9%/99% confidence intervals (close-up of previous figure)

Figure 6: Variations of inefficiency ratios with LDGM Staircase/Triangle, with object size = 20000 packets and FEC ratio = 1.5.

We can see that even with a confidence interval of 99.9% we are quite close to the mean. Yet, by increasing the confidence to 99.99%, the upper confidence interval bound starts to become important (1.187 for LDGM Triangle). The absolute maximum is 1.405 for LDGM Triangle and 1.348 for LDGM Staircase. In spite of small differences, LDGM Staircase and LDGM Triangle show the same global behavior.

We cannot give an explanation for these peaks yet. This will be considered in future works.

3.5 Encoding/Decoding Time Experiments

We now evaluate the encoding/decoding times and the associated bandwidths of all codes.

3.5.1 Encoding/Decoding Times with a Fixed Object Size and Fixed FEC Ratio

As already pointed out, the encoding times for LDPC are very high. For instance the encoding of a 20,000 packet object with a FEC ratio of 1.5 requires 281 seconds on a Pentium IV 3.06 GHz machine (figure 7). Note that our implementation is absolutely not optimized for LDPC encoding, and major improvements will probably be observed by using one of the optimizations suggested in the literature. But in all cases, the performances will remain behind that of the various LDGM codes, since these latter enable an immediate encoding, which will never be possible with LDPC.

The decoding time is also very high (175 seconds) because our LDPC codec requires that the G generator matrix be produced at the receiver too for some internal technical reason[9]. This limitation could be removed in the future, and decoding would then be similar to that observed with all LDGM codes, since the same algorithm is used in all cases. Producing graph 3 (d) took more than a week on a Pentium IV 3.06 GHz with 2GB RAM, whereas tests were repeated only 100 times (compared to 1000 times with LDGM codes). The resulting encoding bandwidth (Table 1) is low, less than 1Mbps.

In comparison all LDGM codes are very fast (e.g. producing graphs 3 (a), (b) and (c) took only one night). There are differences shown in figure 7, essentially caused by the density of the H parity check matrix, which determines the number of XOR sums required to produce each parity packet. For instance, with LDGM Staircase, a parity packet is the sum of only $right\_degree_{LDGM\_Staircase} - 1 = 1 + (3 * k)/(n-k)$ packets with LDGM Staircase, whereas it is the sum of $right\_degree_{LDGM} - 1 = (7*k)/(n-k)$ packets with LDGM. In all cases, the resulting encoding speeds are suitable for transmissions over high speed networks.

Decoding is a bit longer (cf. figure 8) since more manipulations are required (like searching all constraint equations where a given packet takes part and updating the value of the partial sum). However the resulting speeds are also sufficient for high speed network transmissions as show in table 1.

Figure 7 and 8 show that the decoding and especially encoding times for the two RSE variants are significantly slower: LDGM Staircase is about 29.81 times faster during encoding and 13.72 times faster during decoding than RSE/fixed $n = 256$, and respectively 7.44 and 2.96 times faster than RSE/fixed $k = 51$ (table 1). This may become a major issue when working with constrained devices (e.g. PDA), where battery and CPU power is limited.

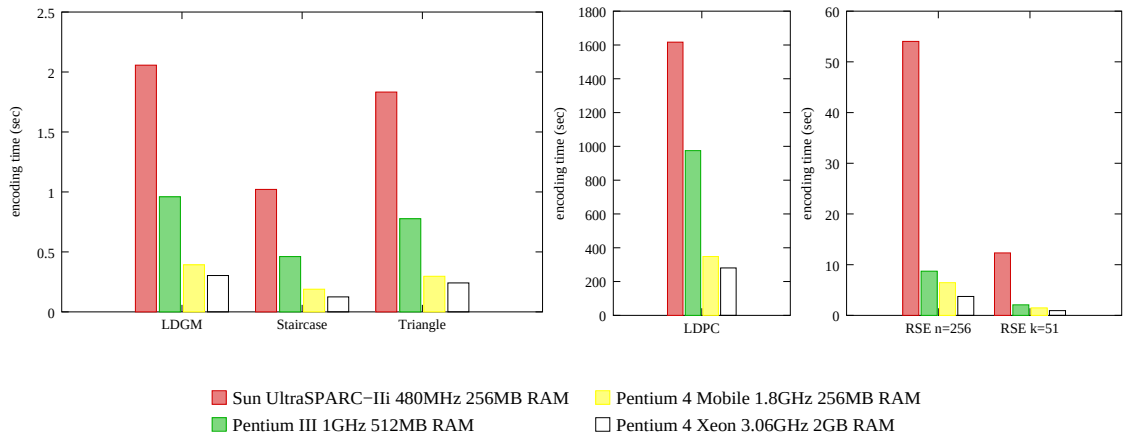


Figure 7: Average initialization plus encoding times for all codes, with object size = 20000 packets and FEC ratio = 1.5.

[9]Matrix columns are permuted by the encoder in order to produce the G matrix, and this permutation must be reproduced at the decoder, which is only possible if G is produced.

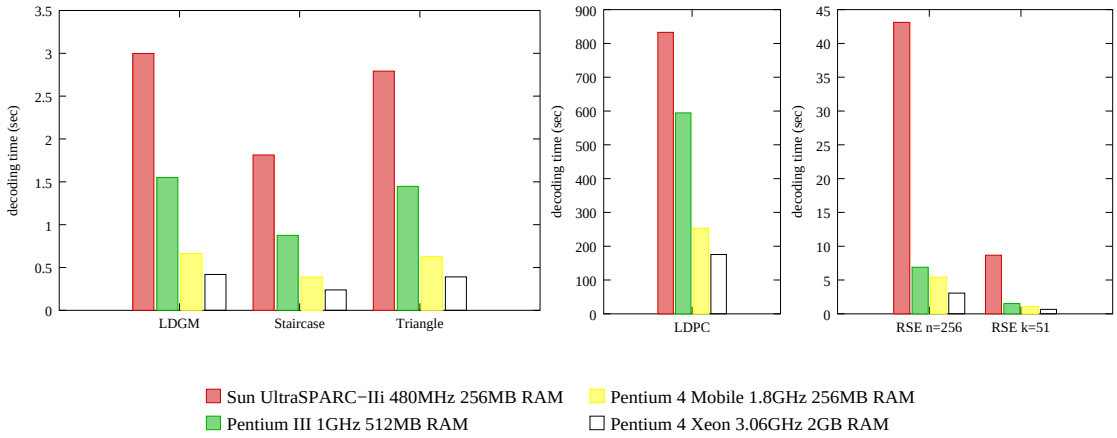


Figure 8: Average initialization plus decoding times for all codes, with object size = 20000 packets and FEC ratio = 1.5.

Code	*Encoding, produced FEC*	*Encoding, produced data + FEC*	*Decoding*
LDGM (left deg. 7)	264.024 Mbps	792.080 Mbps	432.320 Mbps
LDGM Staircase	640.000 Mbps	1,920.000 Mbps	716.773 Mbps
LDGM Triangle	331.952 Mbps	995.848 Mbps	432.826 Mbps
LDPC	0.288 Mbps	0.856 Mbps	0.982 Mbps
RSE (n=256)	21.471 Mbps	64.412 Mbps	52.255 Mbps
RSE (k=51)	86.00 Mbps	257.99 Mbps	241.97 Mbps

Table 1: Average encoding/decoding speeds on a Pentium 4 Xeon 3.06GHz 2GB RAM, with object size = 20000 packets and FEC ratio = 1.5.

3.5.2 Encoding/Decoding Times with Higher FEC Ratios

We now analyze the decoding and encoding times as a function of the FEC expansion ratio, for a fixed object size of $20,000$ packets (figure 9).

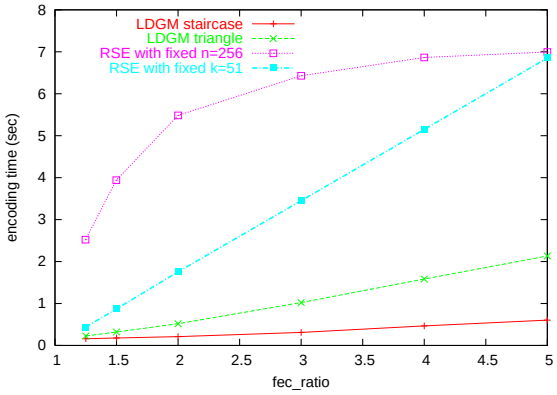
The encoding times (figure 9 (a) and (c)) yield no surprising results. LDGM Staircase remains the fastest codec even with high FEC ratios. The curves remain linear with LDGM Staircase and LDGM Triangle, and the encoding speeds remain acceptable.

Compared to LDGM Staircase, RSE remains quite slow. It is worth noting that RSE/fixed k has a *linear encoding time curve as the object size increases*, unlike common belief. Some results like [2] that show a quadratic encoding time increase, assume that the whole object is encoded as a single block, which requires operating on GF(16) for instance, which is known to yield prohibitive performances, and in no case correspond to the way an RSE codec is used in most applications.

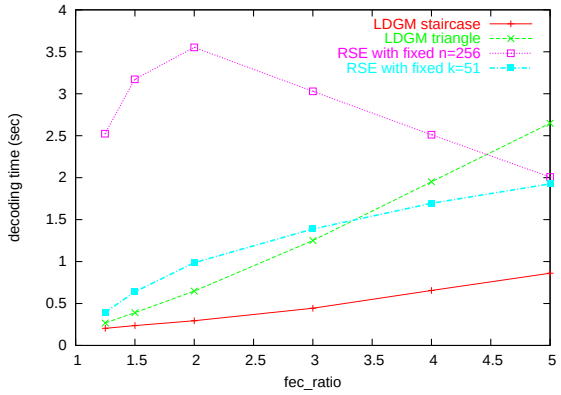
Finally, the LDPC performances (figure 9 (c)) turn out to be catastrophic with higher FEC ratios.

At the decoder (figure 9 (b), (d)), the results are as expected for LDGM Staircase, Triangle and LDPC. LDGM Staircase and Triangle show comparable performances: they are very fast, with a slight advantage for LDGM staircase, and decoding time increases linearly. LDPC show about the same behavior at the decoder than at the encoder because of the limitation already mentioned in our implementation.

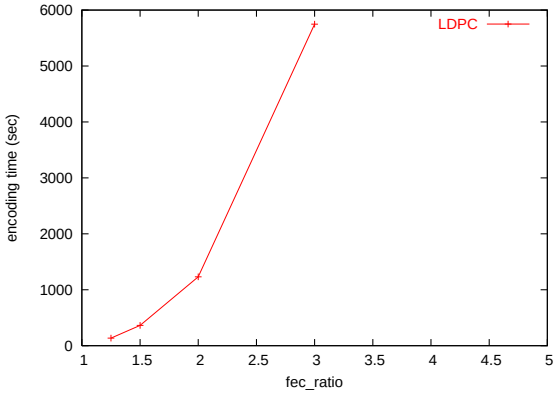
The surprise comes from RSE with a fixed n: decoding time remains reasonable, and even decreases after a maximum at a FEC ratio of 2.0. RSE even becomes faster than LDGM triangle for FEC ratios above ≈ 4.5. The explanation comes from the fact that the block size k of the RSE code is calculated from the maximum value of n divided by the FEC ratio. This means that n decreases with increasing FEC ratio. The number of block increases in the same time, but the time gained having small blocks is more important than the time lost having a higher number of blocks. We also see that with a fixed k



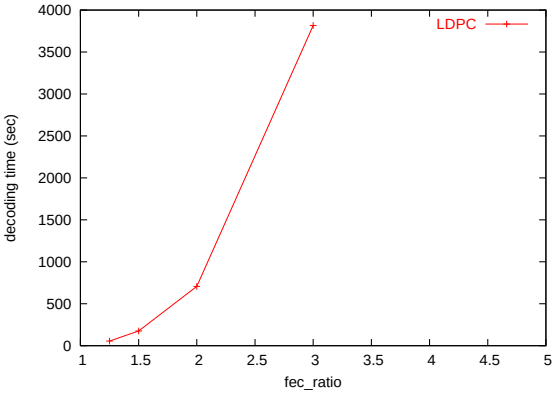
(a) LDGM Staircase/Triangle and RSE encoding times



(b) LDGM Staircase/Triangle and RSE decoding times



(c) LDPC encoding times



(d) LDPC decoding times

Figure 9: Average encoding/decoding times as a function of the FEC expansion ratio on a Pentium 4 Xeon 3.06GHz 2GB RAM, with a fixed object size = 20000 packets.

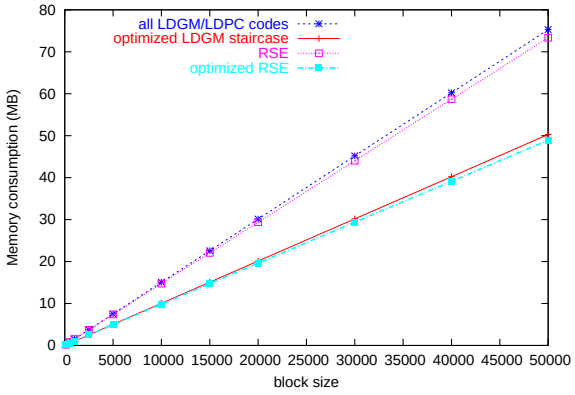
decoding times increases less rapidly than with LDGM triangle. At a FEC ratio of ≈ 3.3 they have the same decoding time.

3.6 Memory Requirements

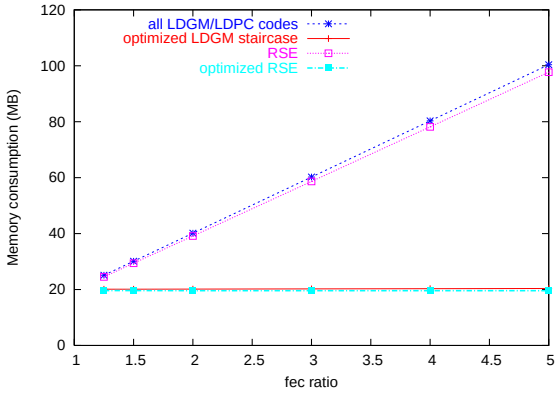
We now analyze the *maximum* memory requirements for the different codecs, both from the encoder and decoder applications points of view. We further distinguish:

- the memory storage of the H matrix (for all LDPC/LDGM codes), plus G in case of LDPC. Note that no such equivalent exists for RSE;
- all the remaining memory requirements, essentially for the source and parity packets (except in optimized versions, see below), the $n - k$ constraint nodes for a decoder (with LDPC/LDGM codes), and also the various buffers or tables required by the codec.

We also consider optimized versions, from a memory requirement point of view, of the encoding or decoding applications:

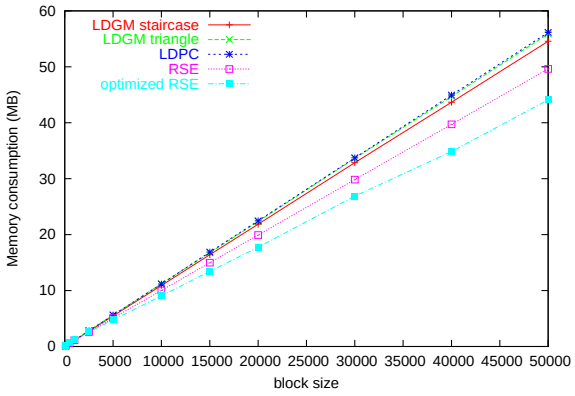


(a) All codes, with a fixed FEC ratio = 1.5.

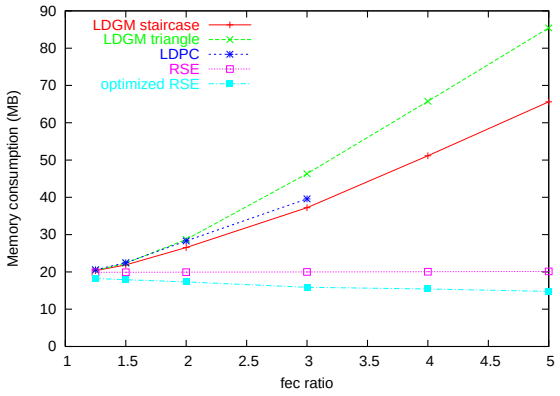


(b) All codes, with a fixed object size = 20000 packets.

Figure 10: Memory consumption (without H/G matrices) during encoding.



(a) All codes, with a fixed FEC ratio = 1.5.



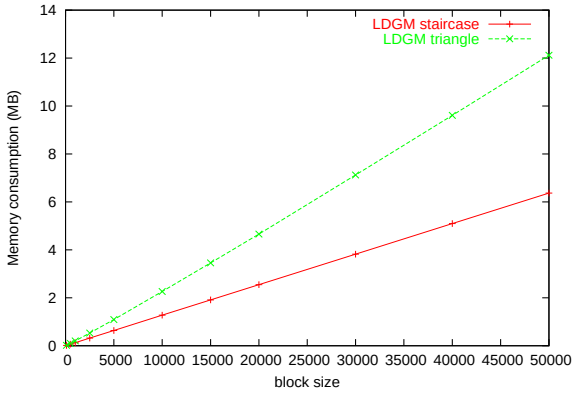
(b) All codes, with a fixed object size = 20000 packets.

Figure 11: Memory consumption (without H/G matrices) during decoding.

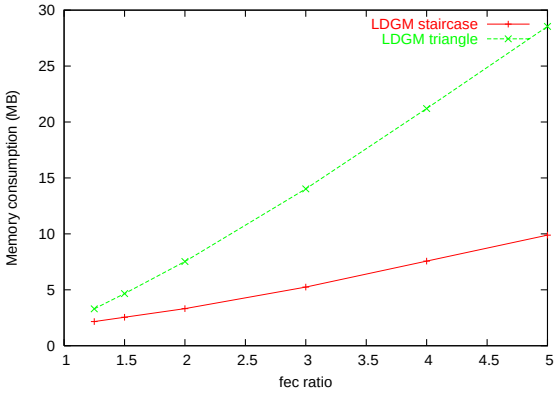
- an optimized encoding application produces a parity packet on the fly, uses it immediately (e.g. by sending it), and then frees the associated packet buffer. This is possible with RSE and LDGM/LDGM Staircase codes, but not with LDGM Triangle where dependencies exist with previously created parity packets.
- in an optimized RSE decoding application, each time a block has been decoded and consumed by the application, the buffers containing the associated packets are deallocated. This optimization assumes that the application can manage each block independently of others, for instance by storing their content on a disk in case of a file transfer [28].

Figures 10 (a) and (b) show the memory requirements at the encoder, not considering the H/G matrices. We see that all non optimized applications (and codecs) have the same requirements, essentially dominated by the amount of data required to store the source and parity packets. The two RSE variants, with fixed n and with fixed k, also have the same memory requirements and are therefore represented with only one curve.

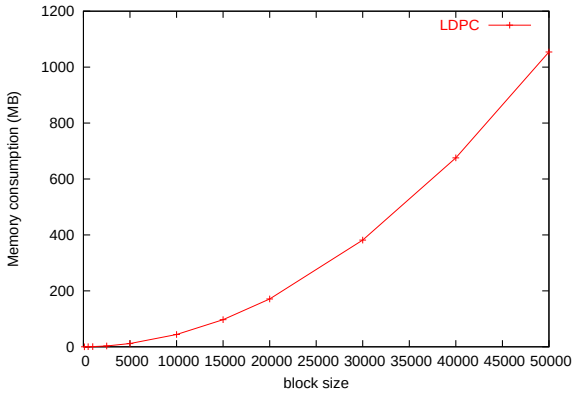
The situation is quite different when considering decoding (figures 11 (a) and (b)). Here RSE (both with fixed n and fixed k) offers significant memory savings, especially with large FEC ratios, even with non optimized applications. The reason is the following: increasing the FEC ratio with RSE does not lead the decoder to store more packets, since a maximum of $k = A\_large$ (or $A\_large - 1$) packets are required for each block. On the opposite, with LDGM codes, there are exactly $n - k$ check nodes in H and therefore $n - k$ partial sums, each of the packet size. These partial sums are required by the decoding process, since they represent the "constant term" of each equation. There is a difference though between the LDGM Staircase and LDGM Triangle codes because the buffers containing these partial sums are not allocated statically (in which case there would not be any difference), but on demand, when an equation is needed (i.e. when a received or decoded packet is implicated in the equation). More equations are used with LDGM Triangle than LDGM Staircase because of the high density of the triangle in matrix H, and therefore more buffers for partial sums are allocated.



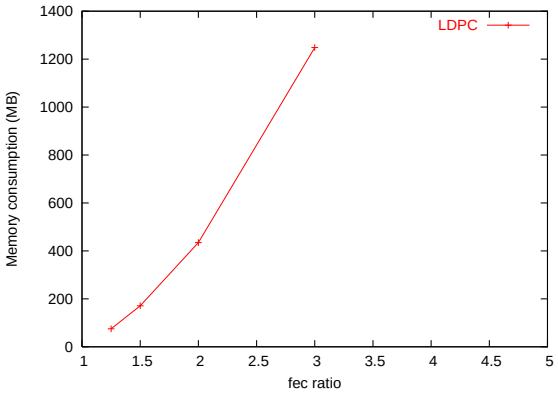
(a) LDGM Staircase/Triangle codes, with a fixed FEC ratio = 1.5.



(b) LDGM Staircase/Triangle codes, with a fixed object size = 20000 packets.



(c) LDPC with a fixed FEC ratio = 1.5.



(d) LDPC with a fixed object size = 20000 packets

Figure 12: Memory consumption for the H matrix (plus G for LDPC).

The memory consumption for the H (plus G with LDPC) matrices is shown in figures 12 (a) to (d). We see that LDPC needs a prohibitive amount of memory for matrix G: 171.12 MB with a block size of $20,000$ packets and a FEC expansion ratio of 1.5. This must be compared to the 2.5 MB required for LDGM Staircase and 4.6 MB for LDGM Triangle for the same parameters. The requirements are

higher when the FEC expansion ratio increases, expecially with LDGM Triangle which requires 28.5 MB with a FEC ratio of 5.0, whereas an LDGM Staircase "only" requires 9.9 MB in that case. Therefore the memory requirements for storing the H matrix, which largely reflect the density of the matrix, is not a major issue with LDGM codes for small to medium FEC expansion ratios, and remains low in front of the memory needed to store packets during encoding.

4 Discussion and Related Works

4.1 Which Code to Use, and When?

The previous performance results clearly define the main field of applications of each code, which, except for LDPC and LDGM, are rather complementary:

- *Regular LDPC showed prohibitive memory requirements and encoding speed*, even if the situation is much better at the decoder[10]. High FEC ratios (> 3.0) could not be supported even on very powerful machines. The measured inefficiency ratio is comparable (but worse) to those obtained with LDGM Staircase/Triangle. We therefore do not recommend the use LDPC which offers no advantage at all.
- *LDGM* is rather limited. It is a fast code but the inefficiency ratios obtained are not satisfying, especially with higher FEC ratios. We therefore do not recommend its use.
- *LDGM Staircase is the best code for FEC expansion ratios* > 2.5 where it outperforms all others codes. In all cases it offers a high encoding/decoding speed (it's the fastest code we tested), along with a low and stable (an important feature) global inefficiency inefficiency ratio.
- *LDGM Triangle is the best code for FEC expansion ratios* ≤ 2.5. In this case it outperforms LDGM Staircase in terms of inefficiency ratio (here also fairly stable) while keeping a good (but lower) encoding/decoding speed. But performances quickly decrease with a FEC ratio > 2.5, in which case the Staircase variant is preferable.
- *RSE* remains an excellent choice for small objects, or when the maximum memory requirements at the decoding side must be kept low, especially with large FEC expansion ratios. But its major limitations are (1) its high encoding (and often decoding) time (and CPU load) compared to LDGM Staircase/Triangle codes, an issue with lightweight hosts, and (2) a global inefficiency ratio much lower than those achieved with LDGM Staircase/Triangle codes.

With all four large block FEC codes, decoding is always fast, but less than encoding, because of more complex operations involved that require to cross the H matrix and store intermediate sum values in check nodes. The performance level is nevertheless compatible with high transmission rates, in the order of several hundreds of Mbps.

4.2 Comparison With Other Fixed-Rate Codes

We now give some elements of comparison with several fixed rate codes. [23] gives recent performance evaluations of various codes derived from LDPC: Tornado-like codes, LDPC codes and IRA codes. In each case the authors only consider *patented irregular graphs*. Because many different degree distributions are possible, the authors systematically test all distributions mentioned in the literature (80 such distributions are considered), and only report the best results. A comparison with our results highlights that:

- using irregular distributions is for sure effective, on condition the right distribution is used. For instance, with a $20,000$ object and an FEC expansion ratio of 1.5, the authors achieve an approximate 1.02 decoding inefficiency with all codes, whereas our best code only achieves 1.055.

[10] We ignore the limitation mentioned in section 3.5 since this can probably be solved –even if we did not try– in an optimized implementation.

- the results obtained with irregular distributions still improve when the object size increases, falling below 1.01 with IRA codes and objects of size 100,000 packets, while in our case an equilibrium is achieved around 1.055.

In [2], using the Tornado Z codec, the authors mention a decoding inefficiency on the average of 1.054 with a FEC expansion ratio of 2, which is better (but not so far) than our 1.087 value. Yet this is achieved with a more complex code, that uses several cascaded bipartite graphs and a final Cauchy codec that protects the last level. Our LDGM codes are in comparison rather simple.

Since no public implementation of either the codes used in [23] or Tornado Z is available, only a limited comparison is possible. Therefore we can not compare such critical aspects as the encoding/decoding speeds, the maximum memory requirements, and the operational complexity (e.g. from [23] it is not sure that the same degree distribution always achieves the best results when varying the object size and FEC expansion ratio, which could create practical problems). If our results are lower than those achieved with the above patented codes, (1) they are not ridiculous though, and (2) the overall codec complexity is significantly lower.

4.3 Comparison with Rate-Less Codes

Fixed rate large block codes are known to have limitations, in particular when compared to rate-less codes[18]. This is essentially caused by the necessity to create and store on both ends (coder and decoder) the same parity check matrix. More specifically:

- *n is predefined:* The source must define beforehand the maximum number of parity packets that can be generated. There is no way to increase this number afterwards, except by generating a new parity check matrix H_2 and starting all over again. But the global efficiency in that case remains limited since H and H_2 are totally unrelated. On the opposite a rate-less code avoids this problem and produces dynamically new parity packets as the need arises.

  This is in our opinion the main limitation of the LDGM Staircase/Triangle codes for some applications, but not all of them (e.g. this is probably not an issue for some distributed storage schemes where the number of replica is decided beforehand).

- *Parity check matrix storage requirements:* the LDGM Staircase/Triangle codes require that both ends store the same H matrix of size $(n-k) \times n$. On the opposite rate-less codes make no use of a parity check matrix. For instance, with the Online code, the packet index itself (a large 128 bit number) is a seed that defines the set of source packets it is the sum.

  Yet our experiments have shown that the amount of memory required to store the H matrix with the LDGM Staircase code (and to a lesser extent the LDGM Triangle code) is relatively low compared the amount of memory required to store source packets, even if it regularly increases with the FEC ratio. This small size is made possible by the sparse property of H which enables the use of efficient data structures (section 3.1). We therefore do not consider this point to be a practical issue with small to medium FEC expansion ratios, that is to say the situations where LDGM Staircase/Triangle codes perform the best.

- *parity packet storage at the encoder:* LDPC and LDGM Triangle codes require that the encoder stores all the parity packets produced since they may be used for producing further parity packets. Yet no such requirement exists with LDGM Staircase (except for the previously produced parity packet), which is an advantage when producing parity packets on-the-fly is sufficient. This situation is in that case similar to that of rate-less codes where parity packets are exclusively the sum of source packets.

- *n, or equivalently the FEC expansion ratio, are limited by practical considerations:* the parity packet storage requirements (and to a lesser extent the H matrix storage requirements) limit the the FEC expansion ratio. Besides, the inefficiency ratio also increases when the FEC ratio increases. There is therefore a incentive to use non over-estimated FEC expansion ratios.

In spite of these limitations, LDGM Staircase/Triangle codes are good codes with many potential applications. These codes are highly recommended when there is no need to produce high numbers of parity packets, while working on large objects (as in distributed storage applications, section 1.3). LDPC codes (in a broad sense, which includes LDGM codes) have also recently been adopted in the DVB-S2 standard, as a replacement of RSE, which is a sign of their potential. Otherwise, in environments where the FEC expansion ratio needs to be high, rate-less codes are clearly preferable. Such codes will be considered in future works.

5 Conclusions

In this paper we describe the rationals behind four large block FEC codes, derived from LDPC, and capable of working on source blocks composed of several tens of thousands of packets. We provide an extensive performance analysis under various situations, and compare them to a widely used Reed-Solomon small block FEC codec.

This work is based on a real codec, distributed under an open-source GNU/LGPL license [21]. Therefore, unlike many other works in the coding theory area, this paper deliberately skips theoretical aspects to focus on practical results. This standpoint enables us to explore several variations of well-known codes, which were mentioned in the literature (e.g. [16] mentions LDGM Staircase codes) but have never been subject to any theoretical analysis, in parts because of the associated complexity.

We try to provide an exhaustive survey of the performances of these codes to fully understand their benefits and limitations, in order to use them in the most appropriate way. We show that the Staircase/Triangle codes have encoding/decoding bandwidths of several hundreds of Mbps, which makes them appropriate to communications over high speed networks, and that they are nicely complementary, depending on the desired FEC expansion ratio.

Another interesting result is the fact that a regular LDPC code has no interest at all, since it provides no erasure protection benefit compared to the LDGM Staircase/Triangle codes while having prohibitive encoding times (even if our codec could be improved from this point of view) and memory requirements.

Our results, in terms of decoding inefficiency, are not ridiculous when compared to previously published large block FEC performances, using patented codes, even if they remain lower. We deliberately choose not to consider techniques which are known to be patented, no matter how efficient they may be. This choice stems from our desire to design patent-free, open source FEC codecs. In particular we do not consider graphs with irregular left/right degree distributions, a technique known to be highly efficient (even if defining an appropriate degree distribution remains a complex task).

Our conclusions is that the LDGM Staircase/Triangle codes have a high potential and can be advantageously used in many situations. Future works will extend the comparison with rate-less codes (e.g. the patent-free Online code [18]) and on new MDS codes like [9] that are capable to work on block sizes significantly higher than with traditional MDS codes.

6 Acknowledgments

The authors would like to warmly thank Julien Labouré who designed the initial LDPC/LDGM codec, and Zainab Khallouf for preliminary contributions on the subject. The author also thank Radford Neal and Jerome Lacan for fruitful discussions during the early stage of this work.

References

[1] Y. Birk and D. Crupicoff. A multicast transmission schedule for scalable multi-rate distribution of bulk data using non-scalable erasure correcting codes. In *IEEE INFOCOM'03*, Mar. 2003.

[2] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *ACM SIGCOMM'98*, Aug. 1998.

[3] H. Ernst, L. Sartorello, and S. Scalise. Transport layer coding for the land mobile satellite channel. In *59th IEEE Vehicular Technology Conference (VTC'04), Milan, Italy*, May 2004.

[4] R. G. Gallager. Low density parity check codes. In *PhD thesis, Massachussets Institute of Technology*, 1960.

[5] R. G. Gallager. Low density parity check codes. *IEEE Transactions on Information Theory*, 8(1), Jan. 1962.

[6] J. Gemmell, E. Schooler, and J. Gray. Fcast multicast file distribution. *IEEE Network*, 14(1), Jan. 2000.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Ninth International ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, Nov. 2000.

[8] J. L. L. Dairaine, L. Lancérica. Enhancing peer to peer parallel data access with peerfect. In *Fifth International Workshop on Networked Group Communication (NGC'03), Munich, Germany*, Sept. 2003.

[9] J. Lacan and J. Fimes. A construction of matrices with no singular square submatrices. In *7th International Conference on Finite Fields and Applications*, May 2003.

[10] M. Luby. Lt codes. In *43rd IEEE Symposium on Foundations in Computer Science*, Nov. 2002.

[11] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. *Asynchronous Layered Coding (ALC) protocol instantiation*, Dec. 2002. IETF Request for Comments, RFC3450.

[12] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, M. Handley, and J. Crowcroft. *Layered Coding Transport (LCT) building block*, Dec. 2002. IETF Request for Comments, RFC3451.

[13] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Improved low-density codes using irregular graphs. *IEEE Transactions on Information Theory*, 47(2), Feb. 2001.

[14] M. Luby, A. Shokrollahi, V. Stemann, M. Mitzenmacher, and D. Spielman. *Irregularly graphed encoding technique*, June 2000. U.S. Patent No. 6,081,909.

[15] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft. *The use of Forward Error Correction (FEC) in reliable multicast*, Dec. 2002. IETF Request for Comments, RFC3453.

[16] D. MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, ISBN: 0521642981, 2003.

[17] D. MacKay and R. Neal. Good codes based on very sparse matrices. In *5th IAM Conference: Cryptography and Coding, LNCS No. 1025*, 1995.

[18] P. Maymounkov. Online codes. Research Report TR2002-833, New York University, Nov. 2002.

[19] P. Maymounkov and D. Mazières. A construction of matrices with no singular square submatrices. In *IPTPS'03*, Feb. 2003.

[20] R. Neal. *Software for Low Density Parity Check (LDPC) codes.* http://www.cs.toronto.edu/∼radford/ldpc.software.html.

[21] C. Neumann, V. Roca, J. Labouré, and Z. Khallouf. *An Open-Source Implementation of a Low Density Parity Check (LDPC) Large Block FEC Code.* URL: http://www.inrialpes.fr/planete/people/roca/mcl/.

[22] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh. *FLUTE - File Delivery over Unidirectional Transport*, Dec. 2003. Work in Progress: <draft-ietf-rmt-flute-07.txt>.

[23] J. Plank and M. Thomason. On the practical use of ldpc erasure codes for distributed storage applications. Research Report UT-CS-03-510, University of Tennessee, Sept. 2003.

[24] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), Apr. 1997.

[25] V. Roca and al. *MCLv3: an Open Source GNU/GPL Implementation of the ALC and NORM Reliable Multicast Protocols.* URL: http://www.inrialpes.fr/planete/people/roca/mcl/.

[26] V. Roca, Z. Khallouf, and J. Laboure. Design and evaluation of a low density generator matrix (ldgm) large block fec codec. In *Fifth International Workshop on Networked Group Communication (NGC'03), Munich, Germany*, Sept. 2003.

[27] V. Roca and B. Mordelet. Design of a multicast file transfer tool on top of alc. In *7th IEEE Symposium on Computers and Communications (ISCC'02), Toarmina, Italy*, July 2002.

[28] V. Roca and B. Mordelet. Improving the efficiency of a multicast file transfer tool based on alc. Research Report 4411, INRIA, Mar. 2002.

[29] A. Shokrollahi. Raptor codes. Research Report DR2003-06-001, Digital Fountain, 2003.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Futurs : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399