Design and Evaluation of a Low Density Generator Matrix (LDGM) Large Block FEC Codec

Vincent ROCA, Zainab KHALLOUF, and Julien LABOURE

INRIA Rhône-Alpes, Planète project, France
{firstname.name}@inrialpes.fr

Abstract. Traditional small block Forward Error Correction (FEC) codes, like the Reed-Solomon erasure (RSE) code, are known to raise efficiency problems, in particular when they are applied to the Asynchronous Layered Coding (ALC) reliable multicast protocol. In this paper we describe the design of a simple large block *Low Density Generator Matrix* (LDGM) codec, a particular case of LDPC code, which is capable of operating on source blocks that are several tens of megabytes long. We also explain how the iterative decoding feature of LDGM/LDPC can be used to protect a large number of small independent objects during time-limited partially-reliable sessions. We illustrate this feature with an example derived from a video streaming scheme over ALC. We then evaluate our LDGM codec and compare its performances with a well known RSE codec. Tests focus on the global efficiency and on encoding/decoding performances. This paper deliberately skips theoretical aspects to focus on practical results. It shows that LDGM/LDPC open many opportunities in the area of bulk data multicasting.

keywords: FEC, large block FEC codes, LDGM, LDPC, reliable multicast, ALC

1 Introduction to the 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: *Automatic Repeat Request (ARQ)*, where lost packets are retransmitted, and *Forward Error Correction (FEC)*, which transmits redundant 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 k *source packets* (A.K.A. original or data packets) are encoded into n packets. If the FEC encoder keeps the k source packets in the set of n packets, this code is called *systematic*. The additional $n - k$ packets are then called *parity packets* (A.K.A. FEC or redundant packets). A receiver can then recover the k source packets provided it receives any k (or a bit more with

LDGM/LDPC codes) packets out of the n possible. The great advantage of using FEC with multicast transmissions is that the same parity packet can recover different lost packets at different receivers. Therefore, the feedback from the receivers can be implemented at a coarser granularity. Using FEC also limits the risk of feedback implosion at the source. An extreme case is the *Asynchronous Layered Coding* (ALC) massively scalable reliable multicast protocol [6, 7] which achieves reliability thanks to an intensive use of FEC. There is absolutely no feedback information from the receivers in that case.

The principal motivation for this work is our *Multicast Library (MCLv3)* [17] which implements the ALC protocol family and provides an integrated solution for the reliable and highly scalable multicast delivery of bulk data. Previous work [1, 9] and the experienced gained with MCLv3 [19, 18] have highlighted the importance of the two parameters (k, n) in the global efficiency achievable at application level:

- *a large n/k ratio is beneficial because it reduces the probability of packet duplication at a receiver.* Indeed, since the same set of packets (data or parity) is transmitted in a different random order in each ALC layers [19], the probability that a receiver receives the same packet on two different layers decreases if the number of packets increases.
- *a large k is beneficial because it increases the correction capacity of the FEC code.* Indeed, a parity packet can only recover an erasure in the block it belongs to. Therefore the erasure capabilities of a parity packet are inversely proportional to the number of blocks a file must be segmented into because of limitations on the k parameter. This is known as the "coupon collector problem" (section 2.5). Ideally a file may be encoded in a single block, in which case each parity packet is useful.

A Reed-Solomon erasure code (RSE) based on Vandermonde matrices [16] is intrinsically limited by the Galois Field it uses. A typical example is GF(2^8) where n is at most equal to 256. With one kilobyte packets, a FEC codec producing as many parity packets as data packets (i.e. $n = 2k$) operates on blocks of size 128 kilobytes at most, and all files exceeding this threshold must be segmented into several blocks. Another drawback of RSE is a huge encoding/decoding time with large (k, n) values, which is the reason why GF(2^8) is preferred in spite of its limitations on the block size. Yet RSE is optimal because a receiver can recover erasures as soon as it has received *exactly* k packets out of n. A FEC code with this property is called *MDS, or Minimum Distance Separation.* Other small block codes exist (e.g. Reed-Muller code) that all share the same fundamental limitation on the (k, n) parameters.

In this paper we describe the implementation details and performances of a large block code: Low Density Generator Matrix (LDGM), which is a particular case of Low Density Parity Check (LDPC) codes [2]. LDGM/LDPC have two main advantages: (1) they enable high speed encoding/decoding, and (2) they operate on large blocks ($k \gg 1$).

The paper is structured as follows: Section 2 gives a brief introduction to the LDGM/LDPC codes and to iterative decoding systems. Section 3 explains how

these codes can be used for the time-limited partially-reliable transmission of a large number of small objects. Section 4 describes experimental results obtained with our LDGM codec. Finally section 5 introduces related works and section 6 concludes the work.

2 Introduction to the LDPC and LDGM codes

This section gives an overview of the LDPC and LDGM codes. Interested readers can refer to [10, 20] for further details.

2.1 Binary versus Packet, Symmetric versus Erasure Channels

LDPC codes were first introduced by Gallager in 1960 [2, 3] and have been almost completely forgotten until Mackay and Neal rediscovered them in 1995 [11]. 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 in 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 main cause of packet losses is router congestion, and the CRC available in most physical layers guaranties that a packet arriving at its destination and given to IP is error-free.

If LDPC, like many codes, can operate on both types, it is much simpler with a *Packet* (resp. Bit) *Erasure Channels (PEC)* (resp. BEC). This is normal since a PEC carries more information than a BSC (i.e. the packets received are error free). In the remaining of this paper we only consider the PEC case.

2.2 Parity Check Matrix and Bipartite Graph: LDPC versus LDGM

LDPC is a linear block code with a very sparse Parity Check Matrix H (i.e. a matrix containing a large number of 0s), hence its name, "Low Density". The H matrix creates constraints between source and parity packets. Said differently, it creates a system of $n - k$ linear equations of n variables, the source and parity packets.

The H matrix can also be represented as a bipartite graph (figure 1) between left nodes, called *message nodes*, and right nodes, called *check nodes* (A.K.A. constraint nodes). The k source packets form the first k message nodes, while the parity packets form the remaining $n - k$ message nodes. There is an edge in this graph between a source node and a check node only if there is a 1 in the corresponding (column, row) entry of H. The edges represent the constraints between the various message nodes. The associated equations are given in figure 1 along each check node.

In the original LDPC approach, the H matrix, of size $n - k \times n$, and the associated bipartite graph encompass all n left nodes (source and parity nodes).

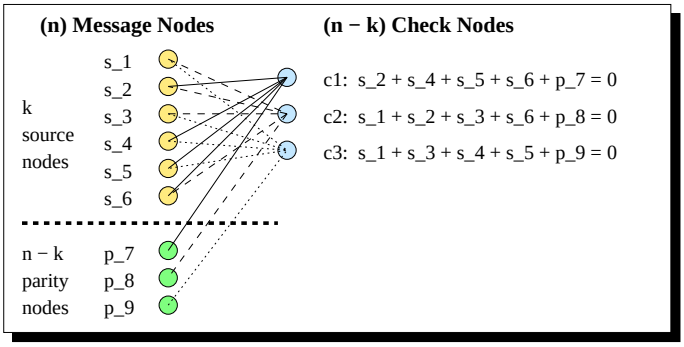

Fig. 1. Example of regular bipartite graph for LDGM, left degree 2, right degree 4.

On the opposite, in the LDGM approach, the H matrix, of size $n-k \times k$ only encompasses the first k source nodes, and is associated to an I_{n-k} identity matrix of size $n-k \times n-k$. Therefore each parity node is linked to exactly one check node. With an LDGM code a dual representation exists, where left nodes are composed only of source packets and right nodes of parity packets [20]. Nevertheless our codec and this paper use the representation of figure 1. The LDPC versus LDGM distinction is important and it impacts:

- *the encoding speed:* encoding is faster with LDGM than with LDPC.
- *the error correction capabilities:* LDPC has a major advantage over LDGM from this point of view, because parity packets are better protected. An LDPC parity packet can be recovered (because it shares edges with several check nodes) which is not possible with LDGM.

In the present paper we will essentially consider the simple case of LDGM codes. Some of the methods presented hereafter will however apply to both codes.

An LDGM code where all source nodes have the same degree, and all check nodes have the same degree, is called *regular*. In that case, the left and right degrees in the bipartite graph are linked to the (k, n) parameters by the equation:

$$k * left\_deg = (n-k) * right\_deg$$

Note that in this equation, the "check node" to "parity node" edges are not considered in the left/right degrees.

The H matrix is essentially constructed randomly, given the left and right degree constraints. An optimization is then applied to remove from the bipartite graph the cycles of length four (i.e. when two source nodes are linked to the same two check nodes) which are known to lead to suboptimal decoding.

2.3 Encoding Process

Encoding is straightforward with an LDGM code: the codec calculates each parity packet as the `XOR` (eXclusive OR) sum of all source nodes associated to

the check node. For instance in figure 1 the parity packet p_7 is the XOR of source packets s_2, s_4, s_5, and s_6.

The encoding is more complex in case of an LDPC code since it requires solving a system of linear equations whose variables are the parity nodes. The generator matrix, G, is this solution (with LDGM H is used as a generator matrix). Encoding just consists in multiplying each source packet by G.

In both cases, since we are working on packets rather than bit streams, the XOR is calculated over all 32 bit words of the packets. Doing it on words rather than bytes is more efficient but requires that the packet length be multiple of 4 (which is not a serious constraint).

2.4 Decoding Process

Decoding Algorithms. Because decoding is exactly the same with LDPC and LDGM, we use both terms indifferently in this section. The decoding over a PEC is straightforward compared to the BSC case, and exploits the following property: *provided the value of all but one message nodes associated to a check node, the missing message node (which can be either a source or a parity packet) is equal to the* XOR *of all other message nodes.* The decoding process consists in solving a system of linear equations using a trivial algorithm:

- if an equation has only one remaining variable, then this variable is equal to the constant term;
- replace this variable by its value in all the other linear equations and reiterate, until blocked or until all variables have been found.

In practice, the set of linear equations are the equations associated to the check nodes, and an incoming packet contains the value of the associated variable. So, each time a fresh packet arrives, we apply the above algorithm and see if decoding can progress by one or more steps. Decoding is successful when all source packets have been received or recovered.

An equivalent algorithm exists and is often presented in the literature [20]: each time a packet arrives at a receiver (or is recovered), its value is XOR'ed to all its neighbor check nodes in the bipartite graph, and the associated edges are removed. Decoding therefore consists in searching a check node of degree one, and if one is found, recovering the missing message node and removing the now useless check node and edges from the bipartite graph. This sequence is one step of the decoding process. Of course, the newly recovered packet can itself trigger other recoveries, so the sequence is repeated recursively.

Practical Aspects. We see that the LDPC decoding process largely differs from an RSE decoding. With RSE, decoding is done all at once, as soon as exactly k packets have been received (MDS property of RSE). With LDPC we only know that decoding is not possible if less than k packets have been received, but there is no way to know in advance how many packets must be received before decoding is successful (LDPC is not an MDS code). Decoding is therefore

performed *step by step*, after each packet arrival. A by-product of this feature is that the CPU load is spread over the whole reception period (even if there are more successful decoding steps at the end), rather than being concentrated at the end with RSE.

Another remark is that encoding and decoding must use the same parity matrix, H. In our codec this matrix is entirely defined once the (k, n, $left\_deg$, *seed*, *H creation algorithm*) parameters are provided, where *seed* is used by the pseudo-random number generator. The source must have a way to communicate explicitly or implicitly these values to the receivers: there can be an exchange of control parameters before transmissions take place, or it can be transmitted in-band using the FEC Object Transmission Information (FTI) header extension of ALC. Another solution is to transmit a compressed version of H to receivers.

2.5 Decoding Inefficiency and Other Practical Sources of Inefficiency

Three sources of inefficiency exist: (1) the FEC decoding inefficiency, (2) the packet duplication inefficiency, and (3) the "coupon collector problem".

LDPC is not an MDS code and introduces a decoding inefficiency: $inef\_ratio * k$ packets ($inef\_ratio \geq 1$) must be received for decoding to be successful. The $inef\_ratio$, experimentally evaluated, is therefore a key performance metric:

$$inef\_ratio = \frac{number\ of\ packets\ required\ for\ decoding}{k}$$

The use of several layers in ALC adds some more inefficiency, because the same set of packets is used for transmissions, in a random order, on the various layers. A receiver may get the same packet from several different layers, thereby creating duplicates. In this paper we will not consider this aspect (for simplicity we assume a single layer is used).

If an object is too large to be encoded in a single LDPC block, it is segmented into several blocks. Because transmissions take place in a random order [19] the "coupon collector problem" occurs [1]: packets are received for blocks already decoded while the receiver waits for the last missing packet of a non-decoded block. This problem is the *main limitation of a small block code like RSE*. Depending on the practical limitations of the LDPC codec (e.g. algorithmic or memory constraints), it can also take place with LDPC, but in a less acute way (the number of blocks is several orders of magnitude lower).

3 Exploiting the Iterative Decoding Process for the Time-Limited Partially Reliable Transmission of a Large Number of Small Objects

We now explain how the iterative decoding process of LDPC can help the partially reliable transmission of a large number of small objects. In this scenario,

transmissions take place in a limited span of time, and each receiver wants to receive and decode as many objects as possible. Even if FEC is used, transmissions are globally pseudo-reliable. Our Scalable Video Streaming over ALC (SVSoA) streaming approach for hierarchically encoded videos [14, 15] typically raises this problem: during each period the source needs to transmit around 1500 objects, each of them being 4.3 KB long, for a total of approximately 6.45 MB.

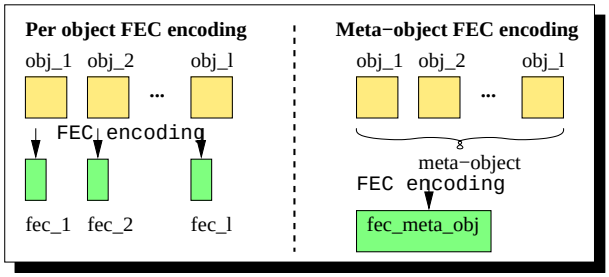

Fig. 2. Per object versus per meta-object FEC encoding.

Inadequacy of RSE. Using RSE requires to encode each object separately, so a FEC packet has only a 1/1500 probability of correcting losses in a given block. We found experimentally that a receiver experiences an average of 85% useless packets to decode all blocks, which is prohibitive! Creating meta-objects and doing cross-object encoding is therefore required to reduce the total number of blocks (figure 2).

But using RSE at meta-object level is not efficient because it creates interdependencies: let's imagine that objects 1 and 2, composed of respectively k_1 and k_2 packets, are gathered in a single meta-object which is globally FEC encoded. Therefore $k_1 + k_2$ distinct packets must be received to decode both objects at once (i.e. the meta-object). A receiver having received only $nb\_rx < k_1 + k_2$ packets cannot decode either of them, even if he received more than k_1 packets for object 1 (with a per-object encoding, decoding of object 1 would be possible then!).

The conclusion is that *RSE is not appropriate, no matter whether it operates on a small object basis or on a meta-object basis.*

The Case of LDPC/LDGM. With an LDPC encoding over the meta-object, a receiver who misses a single source packet of object 1 may take advantage of packets of object 2 because there may be common check nodes in the bipartite graph. The reception of a single packet of object 2 can then trigger the decoding of the missing packet of object 1, even if globally $nb\_rx < k_1 + k_2$. Additionally, the large block feature of LDPC improves the global efficiency since the 6.45 MB meta-object is encoded as a single block.

We found experimentally that the global inefficiency can be reduced to 11.6% additional packets with an LDPC encoding, to compare with the 85% of RSE. *LDPC/LDGM largely outperform RSE for the partially reliable transmissions of large collections of small objects.*

4 Experimental Results

4.1 LDPC Codec Design

We have designed an open-source LDGM codec [5]. This codec uses the R. Neal software [13], which, given the $k, n, left\_degree$ and a seed, creates a regular parity check matrix. This matrix is then used by the coder and decoder parts of our C++ LDGM codec [5]. This codec is implemented as a library that has to be included by another piece of software, either an application or a reliable multicast protocol (e.g. our MCLv3 library implementing ALC [17]). So far our codec assumes that all source and parity packets are stored in physical memory (RAM), which creates limitations for huge blocks Future work will address this aspect and will use the dedicated virtual memory framework included in MCLv3.

4.2 Experimental Setup

The following tests use both our LDGM codec and a popular RSE codec [16]. To k source packets of a block, the codec produces $n - k = k/2$ parity packets (1.5 expansion factor). With RSE we use blocks of at most $k = 64$ source packets. Both source and parity packets are one kilobyte long (the LDGM and RSE codecs both require that all packets be the same size, which means that the application may have to pad the last packet of a block). Once encoding has been performed, the source and parity packets of all blocks are transmitted in a fully random order [4, 19]. Then the receiver waits until it has received enough packets to decode all blocks of the object and then stops reception. There is no explicit loss simulation model, because random transmissions are not affected by the loss model (e.g. random or bursty losses).

During these experiments we measure the encoding/decoding times as well as the minimum number of packets required for decoding. Experiments are repeated 100 times and the minimum/average/maximum values are reported. Note that our transmission scheme leads to a *maximum decoding inefficiency equal to* $n/k = 1.5$ since $n = 1.5 * k$ packets are produced and transmitted.

4.3 Impacts of the Left/Right Degree on Decoding Inefficiency

LDPC performances are known to be highly dependent on the bipartite graph, its regular versus irregular nature, and its left and right degree distribution [12, 8]. In this test we experiment with various degree values. Since we only use regular bipartite graphs, the left degree is the same for all source nodes. The right degree is implicitly defined by the $(k, n, left\_degree)$ T-uple.

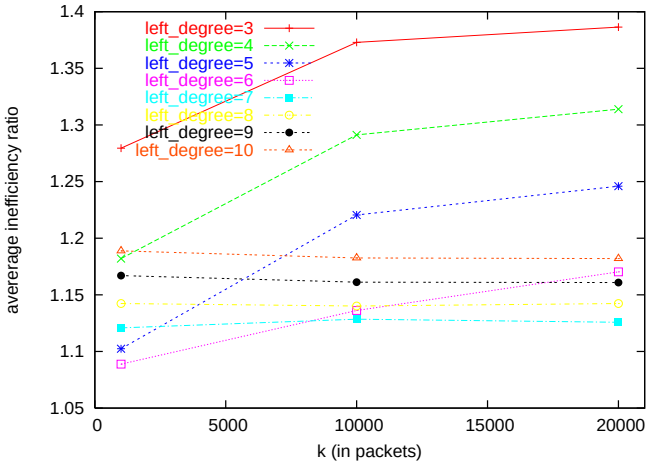

Fig. 3. LDGM left degree comparisons.

Figure 3 illustrates the results obtained. We see that optimal performances are obtained with a left degree equal to 7 when $k \in [8000; 20000]$. More precisely with $k = 10000$ packets and a left degree of 7, the average inefficiency ratio is 1.1284 (said differently there are 12.84% additional packets), the median is 1.1120, the 90% confidence interval around the mean is $+/- 0.0484$, and the inefficiency range is $[1.101; 1.270]$. Note that these results largely differ from that of regular LDPC codes where a left degree equal to 3 is the optimum.

Our results are of course worse than those obtained with highly optimized but proprietary codes [1]. Yet they are encouraging. In the remaining of this paper we only focus on LDGM$(7, 14)$ codes.

4.4 Comparison Between LDGM(7,14) and RSE

We now compare the LDGM$(7, 14)$ and RSE codecs. Results are given in figure 4. We see that LDGM(7,14) largely outperforms RSE for all file sizes. With a 10 MB file, the file is totally decoded with only 12.84% additional packets with LDGM whereas 18.85% additional packets are required with RSE (because of the coupon collector problem, section 2.5). It represents a 31.88% relative improvement.

The average encoding times (PIII-1GHz/Linux) with LDGM(7,14), over a single 10 MB block amounts to 0.171 seconds[1]. This is 14.5 times faster than with RSE (2.486 seconds), where 153 blocks of $k = 64$ source packets each are used[2].

[1] This time does not include the H creation process, which amounts to 0.096 seconds. It is reasonable as this matrix can be reused over several blocks, as long as they all are of the same size.

[2] Note that using a single block of 10000 packets with RSE would lead to prohibitive encoding/decoding times ([1] mentions times of several hours).

Because of technical problems we didn't manage to evaluate the RSE decoding time, yet [16] indicates it is of the same order as the encoding time. The average LDGM decoding time amounts to 0.782 seconds. It is 4.6 times higher than the LDGM encoding time, but still 3.2 times faster than RSE's encoding time (and decoding time with the above assumption).

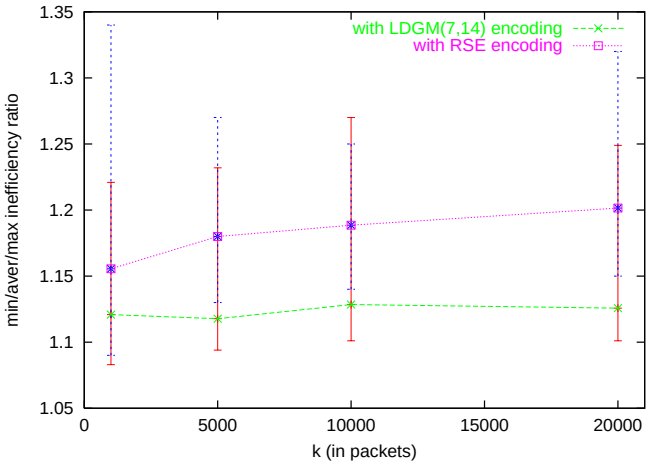

Fig. 4. LDGM(7,14) versus RSE global inefficiency ratio.

5 Related Works

In [16] the theoretical aspects and the design principles of an RSE codec are discussed. The associated open-source implementation turned out to be the most popular FEC codec in the reliable multicast community. This codec can be used either in a GF(2^8), which provides good performances but is limited to $n \leq 255$, or in a GF(2^16) which relieves the (k, n) constraint at the expense of major encoding/decoding times.

[1, 8] introduced the basis of new FEC codes and how they can be used to create a reliable multicast protocol (which lead to the design of the ALC [6] protocol family). This code, Tornado-Z, has performances that are several orders of magnitude higher than that of Reed-Solomon FEC codes, but this is a proprietary code (see [9] for the list of patents). In fact Tornado belongs to the same category as LDPC: codes on graphs, that have been introduced in the 1960's by Gallager [2].

[9] introduces expandable codes, that can produce an infinite number of parity packets ($n \gg 1$) while operating on large blocks. Yet very little is known on the theory behind them, and just like Tornado, the LT code that falls in this category is largely protected by patents.

A lot of theoretical work has been done with LDPC (e.g. [20, 12]). But to the best of our knowledge we are not aware of any public LDPC implementation for the PEC (R. Neal's LDPC software [13] is a simulator, not a codec for the PEC). The present work tries to fulfill this gap.

6 Conclusions

In this paper we describe the design of a simple large block LDGM codec, a particular case of LDPC codes, which is capable of operating on source blocks that are several tens of megabytes long. This is a major asset over the widespread Reed-Solomon codec which is limited to source blocks that are several orders of magnitude smaller, thereby creating important efficiency problems.

We also explain how the iterative decoding feature of LDGM can be used to protect a large number of small independent objects during time-limited partially-reliable transmissions. LDPC/LDGM enable huge performance improvements compared to RSE which is definitely not appropriate.

This paper provides several experimental results that assess LDGM's benefits. Being capable of operating on large blocks largely compensates the intrinsic decoding inefficiency of LDGM which is not an MDS code (unlike RSE). We show that LDGM yields a 31.88% relative improvement compared to RSE with 10 MByte files in terms of global inefficiency. We experimentally found that using a regular matrix with a $(7, 14)$ left/right degree distribution is the optimum in our case. We experienced an average decoding inefficiency of 12.84% (in a $[10.1\%; 27.0\%]$ range) over a 10 MB block. This is of course higher than previously published performances with highly optimized (but proprietary!) large block FEC codes. We are rather confident that our results can be improved, and at least two directions exist: (1) use LDPC rather than LDGM, and (2) use irregular graphs.

Finally encoding and decoding are considerably faster than with RSE. Producing 5000 parity packets from a 10 MB block takes 0.171 seconds on a PIII-1GHz/Linux, which represents a $\frac{5\ MB}{0.171} = 239.5$ Mbps encoding rate. Decoding is highly efficient too (0.782 seconds), even if it is slower than encoding.

Last but not least our LDGM codec [5] is distributed under an open-source license at URL: `http://www.inrialpes.fr/planete/people/roca/mcl/`

7 Acknowledgments

The authors would like to thanks Jérome Lacan, Jérome Fimes, Radford Neal, and Ousmane Diouf for fruitful discussions.

References

1. 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.

2. R. G. Gallager. Low density parity check codes. In *PhD thesis, Massachussets Institute of Technology*, 1960.
3. R. G. Gallager. Low density parity check codes. *IEEE Transactions on Information Theory*, 8(1), Jan. 1962.
4. J. Gemmell, E. Schooler, and J. Gray. Fcast multicast file distribution. *IEEE Network*, 14(1), Jan. 2000.
5. J. Labouré, V. Roca, 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/.
6. M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. *Asynchronous Layered Coding (ALC) protocol instantiation*, Dec. 2002. IETF Request for Comments, RFC3450.
7. 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.
8. 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.
9. 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.
10. D. MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, ISBN: 0521642981, 2003.
11. D. MacKay and R. Neal. Good codes based on very sparse matrices. In *5th IAM Conference: Cryptography and Coding, LNCS No. 1025*, 1995.
12. D. MacKay, S. Wilson, and M. Davey. Comparison of constructions of irregular gallager codes. *IEEE Transactions on Communications*, 47(10), Oct. 1998.
13. R. Neal. *Software for Low Density Parity Check (LDPC) codes.* http://www.cs.toronto.edu/∼radford/ldpc.software.html.
14. C. Neumann and V. Roca. Multicast streaming of hierarchical mpeg-4 presentations. In *ACM Multimedia 2002*, Dec. 2002.
15. C. Neumann and V. Roca. Scalable video streaming over alc (svsoa): a solution for the large scale multicast distribution of videos. Research Report 4769, INRIA, Mar. 2003.
16. L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), Apr. 1997.
17. 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/.
18. 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.
19. V. Roca and B. Mordelet. Improving the efficiency of a multicast file transfer tool based on alc. Research Report 4411, INRIA, Mar. 2002.
20. A. Shokrollahi. Codes and graphs. In *STACS 2000 (invited talk), LNCS No. 1770*, pages 1–12, 2000.