A Comparative Study of Rateless Codes for P2P Persistent Storage

Heverson Ribeiro, Emmanuelle Anceaume

A Comparative Study of Rateless Codes for P2P Persistent Storage

Heverson B. Ribeiro[1] and Emmanuelle Anceaume[2]

[1] IRISA / INRIA, Rennes Bretagne-Atlantique, France
heverson.ribeiro@inria.fr
[2] IRISA / CNRS UMR 6074, France
emmanuelle.anceaume@irisa.fr

Abstract. *This paper evaluates the performance of two seminal rateless erasure codes, LT Codes and Online Codes. Their properties make them appropriate for coping with communication channels having an unbounded loss rate. They are therefore very well suited to peer-to-peer systems. This evaluation targets two goals. First, it compares the performance of both codes in different adversarial environments and in different application contexts. Second, it helps understanding how the parameters driving the behavior of the coding impact its complexity. To the best of our knowledge, this is the first comprehensive study facilitating application designers in setting the optimal values for the coding parameters to best fit their P2P context.*

Key words: peer-to-peer, p2p, storage, rateless codes, fountain codes, LT codes, online codes, data persistency.

1 Introduction

Typical applications such as file sharing or large scale distribution of data all have in common the central need for durable access to data. Implementing these applications over a peer-to-peer system allows to potentially benefit from the very large storage space globally provided by the many unused or idle machines connected to the network. In this case, however, great care must be taken since peers can dynamically and freely join or leave the system. This has led for the last few years to the deployment of a rich number of distributed storage solutions. These architectures mainly differ according to their replication scheme and their failure model. For instance, architectures proposed in [10, 13] rely on full replication, the simplest form of redundancy. Here, full copies of the original data are stored at several peers, which guarantees optimal download latency. However, the storage overhead and the bandwidth for storing new replicas when peers leave may be unacceptable, and thus tend to overwhelm the ease of implementation and the low download latency of this replication schema. This has motivated the use of fixed-rate erasure coding as in [2, 5]. Fixed-rate erasure codes such as Reed-Solomon and more recently Tornado codes mainly consist in adding a specific amount of redundancy to the fragmented original data and storing these redundant fragments at multiple peers. The amount of redundancy, computed according to the assumed failure model, guarantees the same level of availability as full replication, but with an overhead of only a fraction of the original data, and coding and decoding operations in linear time (only guaranteed by Tornado codes). Unfortunately, this replication scheme is intrinsically not adapted to unbounded-loss channels in contrast to Rateless codes (also called Fountain codes). As a class of erasure codes, they provide natural resilience to losses, and therefore are fully adapted to dynamic systems. By being rateless, they give rise to the generation of random, and potentially unlimited number of uniquely coded symbols, which make them fully adapted to dynamic systems such as peer-to-peer systems. Two classes of rateless erasure codes exist. Representative of the first class is the LT coding proposed by Luby [6], and representatives of the second one are Online codes proposed by Maymounkov [7] and Raptor codes by Shokrollahi [12]. The latter class differs from the former one by the presence of a pre-coding phase. A data storage architecture based on a compound of Online coding and full replication has been recently proposed in [8]. Previous analyses have shown the benefit of hybrid replication over full replication in terms of data availability, storage overhead, and bandwidth usage [4, 9]. However, for the best of our knowledge, no experimental study comparing the two classes of fountain codes has ever been performed.

The objective of the present work is to provide such a comparison. Specifically, we propose to compare the experimental performance of both LT and Online codes. Note that Raptor codes [12] could have been analysed as a representative of the second class of fountain codes; however because part of their coding process relies on LT codes, we have opted for Online codes. This evaluation seeks not only to compare the performance of both codes in different adversarial environments, and in different application contexts (which is modeled through different size of data), but also to understand the impact of each coding parameter regarding the space and time complexity of the coding process. As such, this work should be considered, for the best of our knowledge, as the first comprehensive guideline that should help application designers to configure these codes with optimal parameters values.

The remainder of this paper is structured as follows: In Section 2 we present the main features of erasure codes. Section 3 presents the experiments performed on LT and Online codes to evaluate their recovery guarantees, their decoding complexity in terms of XOR operations, and their adequacy to varying size of input data. This section is introduced by a brief description of the architecture in which these experiments have been run. Section 4 concludes.

2 Backgroung on Rateless Erasure Codes

As previously said in the introduction, the main advantages of rateless erasure codes over traditional erasure codes are first that the ratio between input and encoded symbols is not fixed which eliminates the need for estimating the loss-rate beforehand, and second that the encoded symbols can be independently generated on-the-fly. The following section emphasizes the main features of both LT and Online codes. An in-depth description can be found in the respective original papers [6, 7].

2.1 Principles of LT Rateless Codes

Coding process The LT coding process [6] consists in partitioning the data item or message to be coded into $k = n/l$ equal size symbols (also called input blocks), where n is the size of the data-item, and l a parameter that is typically chosen based on the packet payload to be transmitted. Each encoded symbol c_i (also called check block) is generated by *(i)* choosing a *degree* d_i from a particular degree distribution (see below), *(ii)* randomly choosing d_i distinct input symbols (called neighbors of c_i) among the k input symbols, and *(iii)* combining the d_i neighbors into a check block c_i by performing a bitwise XOR operation. Note that the degree and the set of neighbors information d_i associated with each check block c_i needs to be known during the decoding process. There are different ways to communicate this information during the coding process. For instance, a deterministic function may be used to define the degree of each check block and then both coder and decoder can apply the same function in order to recreate the information [6]. In the following, any check block cb_i is represented as a pair $\langle c_i, x_i \rangle$, where c_i is the check block generated by combining d_i neighbors and x_i is the set of the d_i combined neighbors. Figure 1(a) shows the LT coding process represented by a Tanner graph [14]. Specifically, the bipartite graph is such that the first set of vertices represents input symbols k_1, k_2 and k_3 and the second set represents the generated check blocks cb_A, cb_B, cb_C, cb_D and cb_E. Using the generation procedure described here above, a potentially infinite number of check blocks can be generated. Later on, we provide the lower bound CB_0 on the number of check blocks that need to be generated in order to guarantee the success of LT coding with high probability.

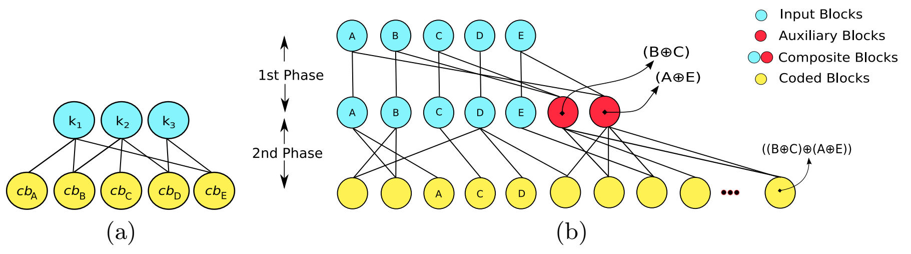


Fig. 1. (a) Check blocks $cb_A = k_1$, $cb_B = k_1 \oplus k_2$, $cb_C = k_2$, $cb_D = k_2 \oplus k_3$, and $cb_E = k_1 \oplus k_3$ coded from input symbols k_1, k_2, and k_3. (b) Online two-phase coding process.

Decoding process The key idea of the decoding process is to rebuild the Tanner graph based on the set of received check blocks. Upon receipt of check blocks, the decoder performs the following iterative procedure: *(i)* Find any check block cb_i with degree equal to one (i.e. each *degree-one* check block cb_i has only one input symbol k_i as neighbor), *(ii)* copy the data c_i of $cb_i\langle c_i, x_i\rangle$ to k_i. (i.e. neighbor k_i of check block cb_i is an exact copy of data c_i), *(iii)* remove the edge between cb_i and k_i, and *(iv)* execute a bitwise XOR operation between k_i and any remaining check block cb_r that has k_i as neighbor ($cb_r = cb_r \oplus k_i$), and remove the edge between cb_r and k_i. These four steps are repeated until all k input symbols are successfully recovered.

Soliton degree distributions The key point of LT codes is the design of a proper degree distribution. The distribution must guarantee the success of the LT process by using first, as few as possible check blocks to ensure minimum redundancy among them and second, an average degree as low as possible to reduce the average number of symbol operations to recover the original data. The first approach proposed by Luby was to rely on the *Ideal Soliton Distribution* inspired by Soliton Waves [11]. The idea behind the Ideal Soliton distribution is that, at each iteration of the decoding algorithm, the expected number of degree-one check blocks is equal to one. Specifically, if we denote by $\rho(d)$ the probability of an encoded symbol to be of degree d ($1 \leq d \leq k$), we have

$$\rho(d) = \begin{cases} \frac{1}{k} & \text{if } d = 1 \\ \frac{1}{d(d-1)} & \text{if } d = 2, \ldots, k \end{cases}$$

Unfortunately, the Ideal Soliton distribution $\rho(.)$ performs poorly in practice since for any small variation on the expected number of degree-one check blocks the recovering is bound to fail. This has led to the *Robust Soliton Distribution*, referred to as $\mu(d)$ in the following. By generating a larger expected number of degree-one check blocks the Robust Soliton distribution guarantees the success of LT decoding with high probability. Specifically, the Robust Soliton distribution is based on three parameters namely k, δ and C, where k is the number of input symbols to be coded, δ is the failure probability of the LT process and C is a positive constant that affects the probability of generating degree-one check blocks. The Robust Soliton distribution $\mu(d)$ is the normalized value of the sum $\rho(d) + \tau(d)$ where $\tau(d)$ is defined as

$$\tau(d) = \begin{cases} \frac{S}{k} \cdot \frac{1}{d} & \text{if } d = 1, \ldots, (\frac{k}{S}) - 1 \\ \frac{S \cdot \ln(\frac{S}{\delta})}{k} & \text{if } d > \frac{k}{S} \\ 0 & \text{if } d = \frac{k}{S} \end{cases}$$

where S, the expected number of degree-one check blocks in the decoding process, is given by $S = C \ln(k/\delta)\sqrt{k}$. Luby [6] proved that by setting the estimated minimum number CB_0 of check blocks to

$$CB_0 = k \cdot \sum_{d=1}^{k} \rho(d) + \tau(d) \tag{1}$$

the input symbols are recovered with probability $1 - \delta$, with δ arbitrarily small.

2.2 Principles of Online Rateless Codes

Coding process The general idea of Online codes [7] is similar to LT codes. The original data is partitioned into k input fragments and coded to redundant check blocks. Online codes are characterized by two main parameters ε and q. Parameter ε, typically satisfying $0.01 \leq \varepsilon \leq 0.1$, infers how many blocks are needed to recover the original message, while q affects the success probability of the decoding process (interesting values of q range from 1 to 5 as shown in the sequel). The main differences between Online and LT codes are the coding algorithm and the degree distribution. Briefly, in contrast to LT codes, Online codes are generated through two encoding phases. In the first phase, $\alpha\varepsilon qn$ auxiliary blocks are generated (typically, α is equal to .55). Each auxiliary block is built by XOR-ing q randomly chosen input blocks. The auxiliary blocks are then appended to the original n blocks message to form the so called composite message of size $n' = n(1 + \alpha\varepsilon q)$, which is suitable for coding. In the second phase, composite blocks are used to create check blocks. Similarly to LT codes, each check block is generated by selecting its degree (i.e. neighbor composite blocks) from a specific degree distribution. Selected neighbors are XOR-ed to form check blocks. Figure 1(b) illustrates these two phases.

Decoding process The decoding process of Online codes performs similarly to LT codes. The composite blocks are decoded and from these decoded blocks, the input blocks are recovered.

Online degree distribution The Online distribution only depends on ε. This parameter is used to calculate an upper bound F on the degree of check blocks with $F = \lceil \ln(\frac{\varepsilon^2}{4}) / \ln(1 - \frac{\varepsilon}{2}) \rceil$. If we denote by $\rho(d)$ the probability of a check block to be of degree d, then the probability distribution $\rho(d)$ is defined as

$$\rho(d) = \begin{cases} 1 - \frac{1+\frac{1}{F}}{1+\varepsilon} & \text{if } d = 1 \\ \frac{[1-\rho(1)]F}{(F-1)d(d-1)} & \text{if } d = 2, \ldots, F \end{cases}$$

F is called the degree sample space of distribution $\rho(d)$. Note that in contrast to LT distribution, the Online distribution $\rho(d)$ does not depend on the number of input blocks. It has been theoretically shown in [7] that generating

$$CB_0 = n(1 + \varepsilon)(1 + \alpha\varepsilon q) \tag{2}$$

check blocks is sufficient to decode a fraction $(1 - \frac{\varepsilon}{2})$ of composite blocks, and to successfully recover the original data with probability $1 - (\frac{\varepsilon}{2})^{q+1}$.

3 Experimental Results

This section is devoted to an in-depth practical comparison of LT and Online codes. This comparison has been achieved by implementing both codes in DataCube, a persistent storage architecture [8]. Prior to comparing both codes performance, we briefly describe the main features of this architecture, and present the different policies that have been implemented to select and collect check blocks at the different peers of the system.

3.1 Experimental Platform

DataCube is a data persistent storage architecture robust against highly dynamic environments and adversarial behaviors. DataCube relies on the properties offered by cluster-based DHTs overlays (e.g. [1, 3]), and by a compound of full replication and rateless erasure coding schemes. Briefly, cluster-based structured overlay networks are such that clusters of peers substitute peers at the vertices of the graph. Size of each cluster is both lower and upper bounded. The lower bound ℓ usually satisfies some constraint based on the assumed failure model. For instance $\ell \geq 4$ allows Byzantine tolerant agreement protocols to be run among these ℓ peers despite the presence of one Byzantine peer. The upper bound L is typically in $\mathcal{O}(logN)$ where N is the current number of peers in the system, to meet scalability requirements. Once a cluster size exceeds L, this cluster `splits` into two smallest clusters, each one populating with the peers that are closer to each other according to some distance D. Similarly, once a cluster undershoots its minimal size ℓ, this cluster `merges` with the closest cluster in its neighborhood. At cluster level, peers are organized as core and spare members. Each data-item is replicated at core members. This replication schema guarantees that in presence of a bounded number of malicious peers, data integrity is guaranteed through Byzantine agreement protocols, and efficient data retrieval is preserved (retrieval is achieved in $\mathcal{O}(logN)$ hops and requires $\mathcal{O}(logN)$ messages, with N the current number of peers in the system). In addition to this replication schema, each data D is fragmented, coded and spread outside its original cluster. The identifier cb_i of each check block i of D is unique and is derived by applying a hash-chain mechanism on the key, *key(D)*, of D such that $cb_i = \mathcal{H}^{(i)}(key(D))$. The adjacencies x_i of cb_i are derived by using a pseudo-random generator function $\mathcal{G}(.)$. The rationale of using $\mathcal{G}(.)$ is that any core member can generate exactly the same check blocks independently from the other core members. Each check block i is then placed at $\gamma \geq 2$ spare members of the cluster that matches cb_i. This coding schema guarantees that in presence of targeted attacks (i.e., the adversary manages to adaptively mount collusion against specific clusters of peers), recovery of the data those clusters were in charge of is self-triggered. For space reasons we do not give any more detail regarding the implementation of hybrid replication in DataCube. None of these details are necessary for the understanding of our present work, as DataCube will essentially be used as an evaluation platform. Anyway, the interested reader is invited to read its description in [8].

To evaluate the coding performance of both Online and LT codes in Datacube, we simulate targeted attacks on a set of clusters. Such attacks prevent any access to the data these clusters cache. By doing this, we force the retrieval of these data to be recovered through the decoding process. Let $\mathcal{C}$ be one these clusters, and D be the data some peer q is looking for. Let $\mathcal{C}'$ be the closest cluster to cluster $\mathcal{C}$ so that, by construction of DataCube, cluster $\mathcal{C}'$ is in charge of recovering cluster $\mathcal{C}$ data, and in particular data D. Then for any peer p_i in the core set of $\mathcal{C}'$, p_i will request and collect sufficiently many check blocks to successfully recover D. Specifically, from the hash-chain mechanism applied to *key(D)*, p_j derives a set M of m check blocks keys, namely $cb_1 \ldots cb_m$, with $cb_i = \mathcal{H}^{(i)}(key(D))$, and $m = m_0 \cdot CB_0$, where $m_0 = 1 \ldots 5$. Then from $\mathcal{G}(.)$ p_j generates $x_1 \ldots x_m$, the respective adjacencies of $cb_1 \ldots cb_m$. From this set M, p_j has the opportunity to apply different policies to select the check blocks it will collect in DataCube, these policies differing according to the priority given to the adjacencies degree. The rational of these policies is to show whether in practice it makes sense to "help" the decoder by first collecting as many degree-one check blocks as possible (this is the purpose of both Policies 2 and 3), or on the contrary, whether it is more efficient to collect random check blocks as theoretically predicted (Role of Policy 1). Policy 4 implements the optimal decoders behaviors. Specifically,

– Policy 1 (*random policy*): no priority is given to the adjacencies degrees. That is, CB_0 check blocks identifiers cb_i are randomly chosen from set M, and the corresponding check blocks are collected in DataCube (through `lookup(`cb_i`)` operations).
– Policy 2 (*degree-one first, random afterwards policy*): the priority is given to degree-one check blocks. That is, all degree-one check blocks identifiers belonging to M are selected, and if less than CB_0 check blocks have been selected, the remaining ones are randomly selected from M. As for above, the corresponding check blocks are collected in DataCube through `lookup(.)` operations.
– Policy 3 (*degree-one-only policy*): similar to Policy 2 except that instead of randomly selected the non degree-one check blocks, degree-two check blocks that will be reduced thanks to the degree-one check blocks are selected. Note that less than CB_0 check blocks can be selected.
– Policy 4 (*optimal policy*): the bipartite graph is applied on the elements of set M, and only necessary check blocks are selected. Comparing to Policy 3, no redundant degree-one check blocks are selected. This makes Policy 4 optimal w.r.t. set M.

If this first phase is not sufficient for the decoding process to recover the input message M, then p_j regenerates a new set M and proceeds as above. It does this until D is fully recovered.

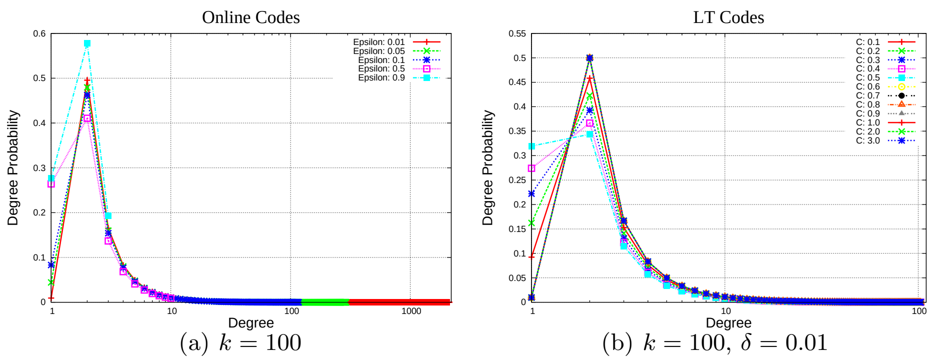
3.2 Setup

Our experiments are conducted over a Linux-based cluster of 60 dedicated Dell PowerEdge 1855 and 1955 computers, with 8 gigabytes of memory each, and 400 Bi-pro Intel Xeon processors. We developed a java-based prototype to simulate DataCube, and to implement both LT and Online coding schemes. Each coding schema is evaluated with a large range of input parameters. The number k of input fragments varies from 100 to $10,000$. For Online coding, ε varies from 0.01

to 0.9, and q is set to integer values from 1 to 5. For LT coding, δ varies from 0.01 to 0.9 and C from 0.1 to 5. All the plotted curves are the average of 50 experiments.

3.3 Degree Distributions

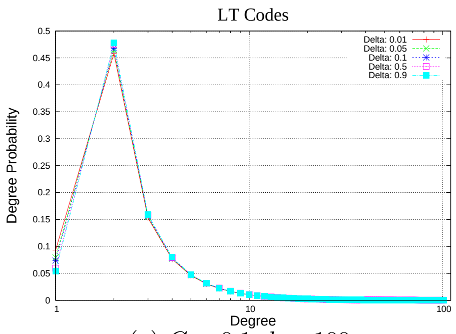
We start by highlighting some differences on the expected behavior of both coding processes. The design of each degree distribution is the most critical point for both coding processes to efficiently recover the original data. A good degree distribution must guarantee the input block coverage, that is, that all the input blocks have at least one edge to guarantee a successful recovery. In Online coding, composite blocks are used to ensure this coverage. Low degrees check blocks, and degree-one in particular, are crucial for the decoding process to start and keep running. On the other hand, too many low degrees may lead to an over redundancy of check blocks, and thus are useless. Figure 2(a) shows the degree distribution of Online codes. Two interesting behaviors can be observed. First, for $\varepsilon \leq 0.1$ the degree-one probability is very small (i.e., ≤ 0.09) while for $\varepsilon > 0.1$, the expected number of degree-one check blocks is greater than 26%. The second observation is that the range of degrees a check block can be built from (i.e., F values) drastically augments with decreasing values of ε (i.e., $F = 3$ for $\varepsilon = 0.9$, and $F = 2114$ for $\varepsilon = 0.01$). As will be shown later on, both features have a great impact on Online coding performance. Figure 2(b) shows the degree distribution



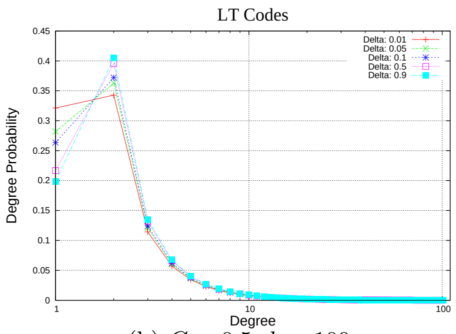
(a) $k = 100$ (b) $k = 100$, $\delta = 0.01$

Fig. 2. Distributions $\mu(.)$ and $\rho(.)$ as a function of the degree.

of the LT coding process. The impact of C on check blocks degrees is clearly shown (i.e., increasing C values augments the probability of degree-one check blocks). An interesting point to observe is that degree-one probability increases up to $C \leq 0.5$, and abruptly drops for $C > 0.5$. Interpretation of this behavior is that combination of these specific values of C (i.e., $C > 0.5$) and those of k and δ lead distribution $\tau(d)$ to tend to zero, which makes the Robust Soliton distribution behaving exactly as the Ideal distribution. The value of C for which this phenomena occurs will be referred to in the following as the *cut-off* value of C (for $k = 100$, and $\delta = 0.01$, the cut-off value equals 0.6). We show in the following how the cut-off value impacts LT performance. Figures 3(a) and 3(b) show the degree distribution of LT codes for varying values of δ. We can see that the influence of δ increases with increasing values of C.



(a) $C = 0.1$, $k = 100$



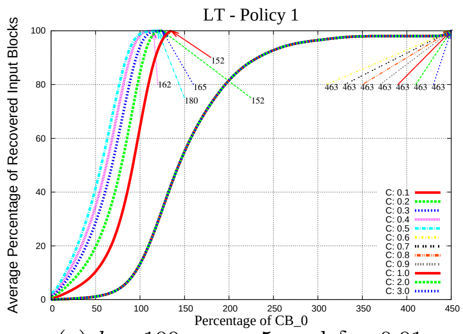
(b) $C = 0.5$, $k = 100$

Fig. 3. Distributions $\mu(.)$ as a function of the degree for different values of δ.

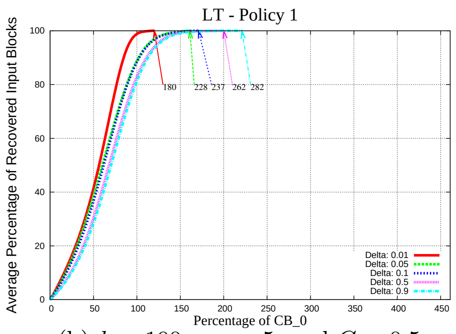
3.4 Recovery Performance of Coding Processes

This section evaluates the number of check blocks that need to be recovered in practice to successfully recover the original data D for both coding schemes, and for the four above described policies. In all the figures shown in this section, curves are depicted as a function of the fraction of the predicted minimal value CB_0. That is, an abscissa equal to 150 means 1.5 times CB_0 check blocks. Arrows point to the number of check blocks that are needed to recover exactly k input blocks, that is 100% of D.

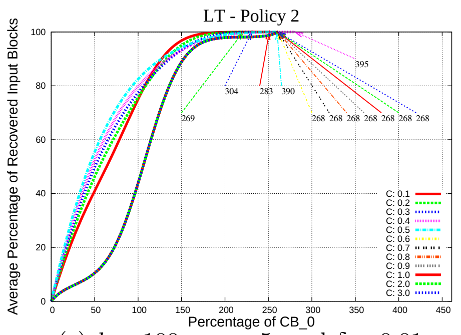
We first analyze the results obtained for LT codes using the four above described policies. The impacts of both C and δ on LT recovery performance when Policy 1 is run are illustrated in Figures 4(a) and 4(b). General observations drawn from Figure 4(a) are first that by randomly collecting check blocks, LT differs from the theoretical prediction in no more than 30% (for instance, to recover $k = 100$ input blocks with $C = 0.1$, CB_0 is equal to 113 (see Relation 1) while in average 152 check blocks are needed. Moreover, for increasing values of C, LT behavior progressively degrades with a sharp breakdown when C reaches its cut-off value (i.e., $C = 0.6$). Indeed, from $C = 0.6$ onwards, the Robust Soliton distribution behaves as the Ideal one, which also explain why LT behavior is independent from C values (i.e., 463 check blocks are necessary to successfully recover $k = 100$ input blocks for any $C \geq 0.6$). Figure 4(b) shows LT recovery performance for different values of δ. We can see that the number of check blocks augments with increasing values of δ. This feature is due to LT distribution $\mu(.)$



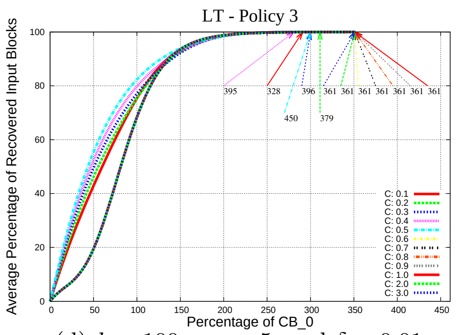
(a) $k = 100$, $m_0 = 5$, and $\delta = 0.01$



(b) $k = 100$, $m_0 = 5$, and $C = 0.5$

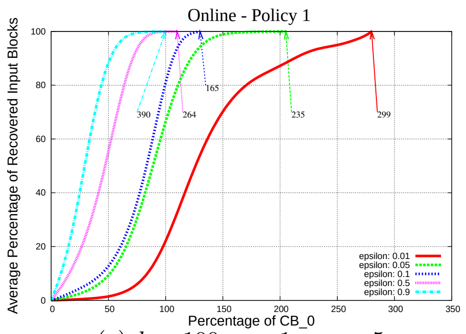


(c) $k = 100$, $m_0 = 5$, and $\delta = 0.01$

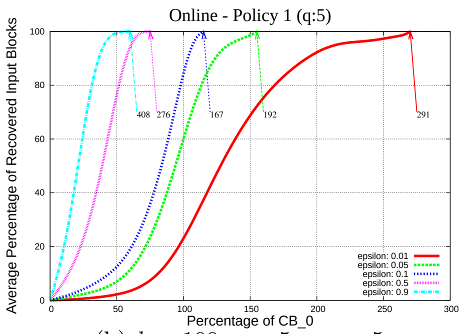


(d) $k = 100$, $m_0 = 5$, and $\delta = 0.01$

Fig. 4. Percentage of recovered input blocks as a function of the number of collected check blocks expressed as a fraction of CB_0 for different values of C.



(a) $k = 100$, $q = 1$, $m_0 = 5$

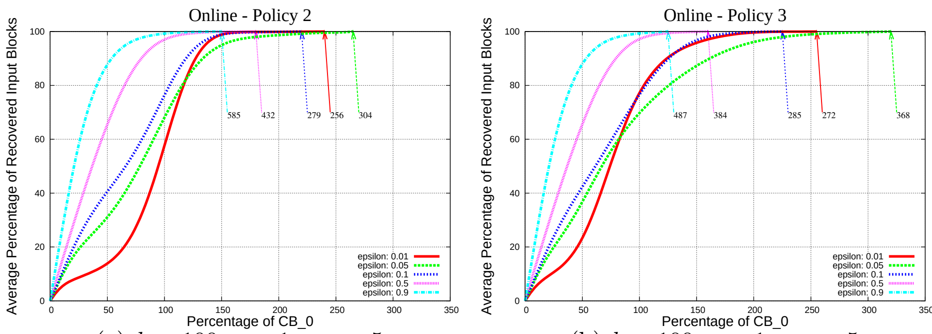


(b) $k = 100$, $q = 5$, $m_0 = 5$

Fig. 5. Percentage of recovered input blocks as a function of the number of collected check blocks expressed as a fraction of CB_0.

since increasing values of δ leads to a diminution of degree-one probability. We can also observe that with 100% of CB_0, the percentage of recovered input blocks increases with decreasing values of δ. A similar behavior observed for different values of C tends to confirm that δ highly impacts the probability of successful recovery. The impact of Policies 2 and 3 on LT is significant as shown in Figures 4(c) and 4(d). The quasi-exclusive collect of degree-one check blocks combined with the impact of C on the generation of degree-one check blocks overwhelms the decoder with too many redundant degree-one check blocks (and degree-two check blocks for Policy 3) which requires many check blocks (from 269 to 395) to successfully recover $k = 100$ original input blocks. On the other hand, when C exceeds its cut-off value, the large amount of degree-one check blocks collected by both policies is compensated by the very low number of check blocks generated by the Ideal distribution, leading to better recovery performance than when $C < 0.6$. Note that in all these experiments, m_0 equals 5 which gives a large choice for the policies to select check blocks that fit their properties. Finally, and as expected, the optimal policy behaves perfectly well. This policy guarantees the full recovery of input blocks in a linear number of check blocks. Indeed, each check block is the outcome of the bipartite graph decoding process, and thus each single selected check block is useful for the recovery (for instance, 103 check blocks in average allow to recover $k = 100$ input blocks. For space reasons we have not illustrated performance of Policy 4 in this paper).

We now analyze the results obtained for Online coding schema with the four policies. Policy 1 is illustrated for different values of ε and q in respectively Figures 5(a) and 5(b). A preliminary observation drawn from Figure 5(a) is that varying ε leads to a greater range of CB_0s values than what is obtained when one varies parameter C in LT (i.e., with Online, CB_0 varies from 106 to 390 while with LT, CB_0 varies from 103 to 150). Thus as a first approximation, we may expect that in average the number of check blocks that need to be collected for successfully recovering the input blocks is larger with Online than with LT.



(a) $k = 100$, $q = 1$, $m_0 = 5$ (b) $k = 100$, $q = 1$, $m_0 = 5$

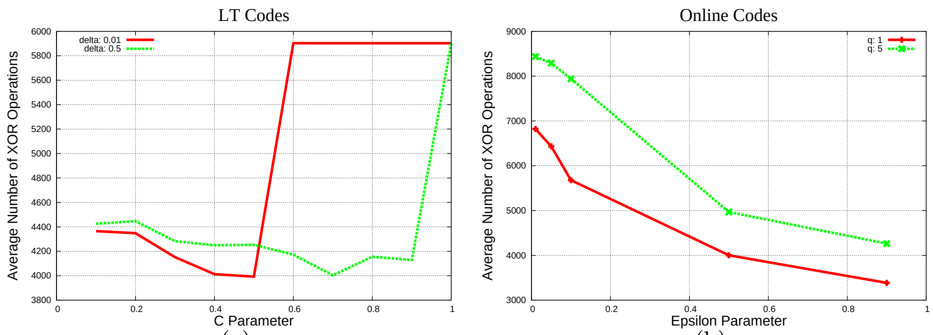
Fig. 6. Percentage of recovered input blocks as a function of the number of collected check blocks expressed as a fraction of CB_0.

Another preliminary comment is that the gap between theoretical predictions and practical results increases with diminishing values of ε. Surprisingly enough, the number of check blocks that need to be collected for a successful recovery does not vary proportionally to ε, that is for extrema values of ε, this number is respectively equal to 290 and 390, while it decreases to reach its minimum 165 for $\varepsilon = 0.1$. Actually, for $\varepsilon = 0.9$, the number of auxiliary blocks is large (see Section 2.2) but the space degree F is very small (i.e., equal to 3, see Section 3.3). Thus a large number of redundant check blocks are *a priori* collected, which demands for more check blocks to successfully recover k input blocks. Now for $\varepsilon = 0.01$, the number of auxiliary blocks is small but they form a complete bipartite graph with their associated input blocks, which clearly make them useless. Moreover the probability of having degree-one check blocks is very small (i.e., 0.01), and the probability of having degree-two is large (i.e., 50%). However as the space degree F is very large too, the likelihood of having some very high degree check blocks is not null, and thus a large number of check blocks need also to be collected. Finally, for $\varepsilon = 0.1$, the distribution of check blocks degrees is relatively well balanced, in the sense that the proportion of degree-one is relatively high (i.e., 9%) but not too high to prevent redundancy. The proportion of degree-two is high (i.e., 47%) which combined with degree-one allows to cover many input blocks. Finally, degree-three and degree-four check blocks are also useful (i.e., respectively equal to 15% and 4%). This clearly make this value of ε optimal, which is confirmed by the fact that 165 check blocks successfully recover 100 input blocks. The very same argument applies for larger values of q as shown in Figure 5(b). The impact of Policies 2 and 3 on Online coding shown in Figures 6(a), and 6(b) is similar to the one obtained on LT, in the sense that recovering essentially degree-one check blocks cumulates with the high proportion of degree-one check blocks generated with the degree distributions (see Section 3.3) and thus, leads to the collect of a large number of redundant check blocks. Note that sensibility of this phenomena augments with increasing values of ε.

3.5 Computational Cost in Terms of xor Operations

In this section, we discuss the computational costs of both LT and Online decoding process. The computational cost is quantified by the number of XOR operations that need to be run to successfully recover the input data when Policy 1 is applied. Note that the unit of XOR used by the encoder/decoder matches the length of an input block.

We first analyze the number of XOR operations in both LT and Online coding schemes as a function of their respective parameters (C, δ), and (ε, q). Results are depicted in Figures 7(a) and 7(b). The main observation drawn from Figure 7(a) is that the number of XOR operations slowly drops down with increasing values of C (provided that C cut-off values are not reached (i.e., $C = 0.6$ for $\delta = 0.01$ and $C = 1$ for $\delta = 0.5$). Specifically, for small values of C, the probability of degree-one check blocks is less than 10% and weakly depends on δ values. On the other hand, for larger values of C, the probability of degree-one check blocks is equal to respectively 22% and 33% for $\delta = 0.5$ and $\delta = 0.01$ which explain the negative slopes of both curves, and the increasing gap between both curves for increasing values of ε.



(a) (b)

Fig. 7. Average number of XOR operations to successfully recover $k = 100$ input blocks as a function of LT and Online parameters.

Now, regarding Online computational cost as a function of ε and q, Figure 7(b) shows first that ε has a strong impact on the number of XOR operations that need to be performed. Specifically for increasing values of ε, this number drastically decreases, down to the average number of XOR operations run with LT coding. This behavior seems to result from the combination of two phenomena. For $\varepsilon = 0.01 \ldots 0.1$, the number of check blocks needed to successfully recover the input data decreases (as previously observed in Figure 5(a)), and in the meantime, F equally decreases with an increasing probability of generating degree-one and degree-two check blocks. Thus both phenomena cumulate and give rise to the diminution of the number of XOR operations as observed in Figure 7(b) (recall that degree-one check blocks do not trigger any XOR operations). Now, for $\varepsilon = 0.1 \ldots 0.9$, the number of check blocks increases (as previously observed in Figure 5(a)), and F drastically decreases together with a high probability of generating degree-one and degree-two check blocks. Thus despite the fact that a large number of check blocks need to be decoded, most of them have a degree-one or degree-two, and thus most of them do not trigger XOR operations, which explains the negative gradients of the curves in Figure 7(b). The second observation drawn from this figure is that the number of XOR operations equally depends on q value. Specifically, for large values of q (e.g., $q = 5$) the number of auxiliary blocks is high and their adjacencies degree with the input blocks is small. Thus these auxiliary blocks are involved in the decoding process and thus augment the number of XOR operations. On the other hand, for small values of q (e.g., $q = 1$), the number of auxiliary blocks is small however, these blocks form a quasi-complete bipartite graph with the input blocks, which makes them useless for the decoding process. Consequently, they do not impact the computational cost of Online. Finally, Figures 8(a) and 8(b) depict the computational cost of both LT and Online as a function of the number of input blocks. The main observation is that both LT and Online decoding complexity are linear with k. The second finding is that for increasing values of k, Online decoding complexity is more sensible to parameters variations than LT is.

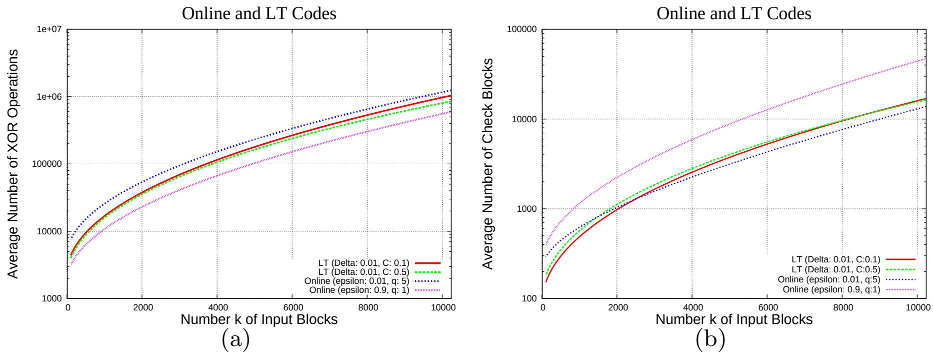


Fig. 8. (a) Average number of XOR operations for respectively LT and Online codes to successfully recover the input data as a function of the size k of the input data. (b) Average number of check blocks for respectively LT and Online codes to successfully recover the input data as a function of the size k of the input data.

4 Conclusion

In the present work, we have evaluated both LT and Online rateless codes in terms of recovery and computational performance, and scalability properties. Experiments have confirmed that it is more efficient to collect random check blocks as theoretically predicted than favoring only small degree check blocks. We expect that this study should allow a good insight into the properties of these codes.

In particular we have confirmed the good behavior of both coders/decoders when the number of input blocks increases. This is of particular interest for multimedia and storage applications.

References

1. E. Anceaume, F. Brasiliero, R. Ludinard, and A. Ravoaja. Peercube: an hypercube-based p2p overlay robust against collusion and churn. In *Proceedings of the IEEE International Conference on Self Autonomous and Self Organising Systems (SASO)*, 2008.
2. R. Bhagwan, K. Tati, Y.C. Cheng, S. Savage, and G.M. Voelker. Total Recall: System support for automated availability management. In *Proceedings of the USENIX Association Conference on Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.
3. A. Fiat, J. Saia, and M. Young. Making chord robust to byzantine attacks. In *Proceedings of the Annual European Symposium on Algorithms (ESA)*, 2005.
4. Y. Houri, M. Jobmann, and T. Fuhrmann. Self-organized data redundancy management for peer-to-peer storage systems. In *Proceedings of the 4th IFIP TC 6 International Workshop on Self-Organizing Systems (IWSOS)*. Springer-Verlag, 2009.
5. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. OceanStore: an architecture for global-scale persistent storage. *ACM SIGARCH Computer Architecture*, pages 190–201, 2000.
6. M. Luby. LT codes. In *Proceedings of the IEEE International Symposium on Foundations of Computer Science (SFCS)*, 2002.
7. P. Maymounkov. Online codes. *Research Report TR2002-833, New York University*, 2002.
8. H.B. Ribeiro and E. Anceaume. DataCube: a P2P persistent storage architecture based on hybrid redundancy schema. In *Proceedings of the IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP)*, 2010.
9. H.B. Ribeiro and E. Anceaume. Exploiting Rateless Coding in Structured Overlays to achieve Data Persistence. *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2010.
10. A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *ACM SIGOPS Operating System Review*, 35(5):188–201, 2001.
11. J.S. Russell. Report on waves. In *14th Meeting of the British Association for the Advancement of Science*, pages 311–390, 1844.
12. A. Shokrollahi. Raptor codes. *IEEE/ACM Transactions on Networking*, pages 2551–2567, 2006.
13. E. Sit, A. Haeberlen, F. Dabek, B.G. Chun, H. Weatherspoon, R. Morris, M.F. Kaashoek, and J. Kubiatowicz. Proactive replication for data durability. In *Proceedings of the 5rd International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.
14. R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.