LT Network Codes: Low Complexity Network Codes

Mary-Luc Champel, Kévin Huguenin, Anne-Marie Kermarrec, Nicolas Le Scouarnec

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

LT Network Codes: Low Complexity Network Codes

Mary-Luc Champel — Kévin Huguenin — Anne-Marie Kermarrec — Nicolas Le Scouarnec

N° 7035

Septembre 2009

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7035--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

LT Network Codes: Low Complexity Network Codes

Mary-Luc Champel*, Kévin Huguenin†, Anne-Marie Kermarrec‡ ,
Nicolas Le Scouarnec*

Thème : Réseaux, systèmes et services, calcul distribué
Équipes-Projets Asap

Rapport de recherche n° 7035 — Septembre 2009 — 23 pages

Abstract: In this paper, we present a new low complexity approach to network coding trading traditional random linear network codes against an extension of LT codes. Network coding is an appealing paradigm in the context of content dissemination as it significantly improves throughput, leveraging path diversity in networks, be they logical or physical. In linear network coding, nodes send linear combinations of packets they have received. However, computing the optimal set of linear functions to apply at each node requires a global knowledge of the network's topology. Random linear network codes (RLNC) address this issue and rely only on a local knowledge of the topology. Yet, decoding requires an $O(n^3)$ Gaussian elimination. Following the observation that randomized network codes have been built upon rateless codes (random linear codes), we explore the feasibility of network coding inspired from another class of rateless codes, namely LT codes, which can be decoded in $O(n \log n)$ operations. We propose a re-encoding method to extend LT codes into new low complexity network codes (LTNC). In P2P dissemination of information, we observe that LTNC trades advantageously communication optimality of RLNC with decoding cost as it incurs only 38.5% of bandwidth overhead for a gain of almost 99% in decoding cost.

Key-words: network codes, low complexity decoding, peer-to-peer

* Thomson R&D
† Université Rennes 1 / IRISA
‡ INRIA Rennes Bretagne Atlantique

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

LT Network Codes : Network Codes à basse complexité

Résumé : Ce document présente une nouvelle technique de codage réseau s'appuyant sur des codes LT. Dans ce document nous définissons une opération de recodage qui permet de construire les LTNC des codes réseaux pouvant etre décodé par propagation de croyance.

Mots-clés : codage réseau, décodage basse complexité, pair-à-pair

1 Introduction

Network coding, initially proposed by Ahlswede *et al.* [1], is a powerful paradigm enabling to significantly improve the throughput in content dissemination applications. Linear network coding paradigm path diversity and relies on each internal node re-encoding content, in the form of linear combinations of packets, instead of simply forwarding data. Since its publication, network coding has generated a growing interest in the community and has been steadily applied to a wide range of applications.

In [3], a survey of network coding and its applications in the Internet and wireless networks is presented and shows the multiple faces of network coding. Its main characteristic is that it allows achieving the maximum throughput on any network, thus enhancing the performance of broadcast operations. Yet, to get the most out of network coding, the complete knowledge of the network topology is required, which obviously limits its applicability in large scale networks, be they logical or physical. To this end, random linear network codes have been proposed where nodes send random linear combinations of received packets. Such an approach does not rely on the knowledge of the whole topology. Random linear network codes have been successfully adopted in file distribution applications over P2P networks. For instance, *Avalanche* [6, 5] uses network coding for file distribution, achieving a throughput 2 or 3 times better than transmitting unencoded packets over *Bittorrent*-like networks [4].

Despite the success of network coding, some works have shown that one of the limitations of the current forms of network coding, namely random linear codes, is that they require a high complexity decoding process. Wang *et al.* [19] compared the performance of network coding in *Avalanche* to systems that do not use coding. One of their main criticisms to network coding is that even if one reduces density in order to reduce the encoding and re-encoding cost, decoding remains costly. Ma *et al.* [12] did another study where they reduced the cost of encoding and re-encoding by using sparse (low density) network codes. They conclude that the high cost of decoding what is received, using Gaussian elimination, still outweighs the benefits obtained. Therefore, they conclude that network coding is not really practical. Despite these controversial works, network coding, in addition to improving the throughput, significantly improves the resilience to failure. This is of the utmost importance in large scale networks subject to churn and dynamicity. Typically, *Avalanche* [6] has a better resilience to churn and to dynamicity than non-coding protocols as long as one can accommodate the high decoding cost.

Clearly, network coding presents many advantages. Yet random linear network codes, such as those used in the *Avalanche* system, still suffer from prohibitive decoding costs. In this paper, we tackle this issue and provide a practical network coding solution built on LT codes. Our LT network codes (LTNC) dramatically reduce the complexity of the decoding process, significantly improving the practicality of network coding in large-scale systems, more specifically in settings where processing power is limited (embedded devices, limited power supply) or when there are time constraints on decoding such as realtime VoD.

In this paper, we build low complexity network codes upon LT codes, namely LT network codes. LT codes can be decoded with a low complexity but cannot generate fresh encoded symbols on the fly before all symbols are decoded. We propose a re-encoding method to combine non-decoded received symbols while

preserving the statistical properties of LT codes. With LTNC, the freshly re-encoded symbols preserve the structure and properties of LT codes to maintain decodability using the low complexity decoding algorithm. We evaluate our LTNC over a P2P content-dissemination application.

In Section 2, we enlight the similarities between rateless codes and network codes. In Section 3, we present our contribution, an extension of low complexity LT codes into new low complexity network codes. In Section 4, we evaluate the performances of LTNC codes by comparing them to RLNC and a dissemination protocol not relying on network coding. In Section 5, we review related works presenting other attempts to build low complexity network codes or other constructions close to LT codes. Section 6 concludes the paper.

2 Background

Network coding is a generic paradigm that can be applied at the router level (physical network topologies) as well as at the application level (overlay topologies). Ahlswede *et al.* showed in [1] that performance can be increased if internal nodes send the result of a function $f(i_1, \ldots, i_n)$ applied to their inputs $\{i_1, \ldots, i_n\}$. This significantly improves the throughput. In Figure 1 max flow cannot be reached if node 3 cannot re-encode data. To this end, nodes must agree on the set of functions f they apply to the data flowing through the network. Finding an optimal set of functions f requires global knowledge on the network topology and involves CPU intensive computation, which significantly limit the practicality of network coding. In order to overcome this issue, randomized schemes have been proposed. They are fully distributed while achieving close to optimal performance.

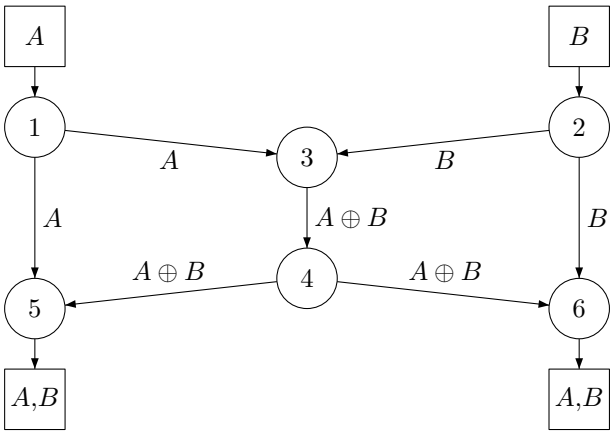


Figure 1: Network coding allows reaching max flow (2.0) on the network. Without network coding at node 3, the max flow would have been 1.5.

In this section, we review randomized schemes for network coding, namely random linear network codes (RLNC). We establish a correlation between randomized schemes for network coding and rateless codes. This observation is at the core of the original network codes proposed in this paper, exhibiting a lower

complexity by using low complexity rateless codes, namely LT codes. We finally present some background on LT codes, required to understand our contribution.

RLNC [8] are fully decentralized, simple and close to optimal network codes, relieving the need for global knowledge of the network's topology. Each node in the network generates fresh encoded symbols by computing random linear combinations of symbols it has received. Recovering the original data at the nodes is achieved by Gaussian elimination on the encoded symbols received.

RLNC can be thought of as an extension of random linear codes. As a matter of fact, random linear codes are traditional erasure-correcting codes. In such a scheme, data is encoded and transmitted only by the source and decoded by the receivers. However, a node cannot generate fresh encoded symbols from encoded symbols unless it can decode them. RLNC are in fact random linear codes enriched with a re-encoding operation. The re-encoding operation consists in computing a random linear combination of encoded symbols. Such newly encoded symbol is in turn a random linear combination of original symbols. For example, suppose $e_1 = o_1 \oplus o_2$ and $e_2 = o_1 \oplus 2o_2$ are re-encoded into $r_1 = e_1 \oplus e_2$, then $r_1 = 2o_1 \oplus 3o_2$ is a fresh random linear combination of original symbols o_1 and o_2. This scheme is close to optimal but the decoding operation remains costly as it involves a Gaussian elimination, the complexity of which is $O(k^3)$.

RLNC are in fact an extension of random linear codes with a re-encoding operation. An interesting approach is to consider random linear codes as rateless codes. Rateless codes [13, 14] allow to generate a virtually infinite set (i.e. q^k linear combinations in a finite field of size q) of encoded symbols from k original symbols. Rateless codes exhibit some interesting properties shared with randomized network codes. Both allow nodes to generate fresh encoded (or re-encoded) symbols independently, without coordination. Both allow decoding and recovering the k original symbols as soon as $k(1 + \varepsilon)$ symbols are received ($\varepsilon \ll 1$).

Stemming from this observation, we propose a new network coding approach that significantly improves the complexity of decoding from $O(k^3)$ to $O(k \log k)$. To this end, we extend LT codes with a re-encoding method, leveraging the low complexity decoding procedure of LT codes. In the rest of this section, we present LT codes. We examine their encoding procedure, their decoding algorithm and the data structure used for decoding, namely Tanner Graphs [17].

Luby proposed Luby Transform codes (LT codes) [11]. These codes are a rateless version of low density parity check codes. They are close to optimal as k original symbols can be recovered from any $k(1+\varepsilon)$ encoded symbols with $\varepsilon \approx 0.05$. Despite the slightly higher overhead, LT codes present many advantages over RL codes. First, as they are simple parity codes, the operations used are performed in $GF(2)$ thus alleviating the need for large multiplication tables and time consuming computations. Moreover, due to their specific structure, they can be decoded by a low complexity algorithm involving only $O(k \log k)$ operations, namely belief propagation.

The encoding process of LT codes is based on linear combinations and involves randomized choices in the selection of the symbols to be combined. It is composed of two steps. First, the degree d of the encoded symbol to generate is chosen according to the Robust Soliton distribution defined in [11] (See Figure 2). The degree of an encoded symbols e is defined as the number of original symbols involved in the linear combination (i.e., XOR) to build the symbol. Second, d original symbols are chosen uniformly and the linear combination of

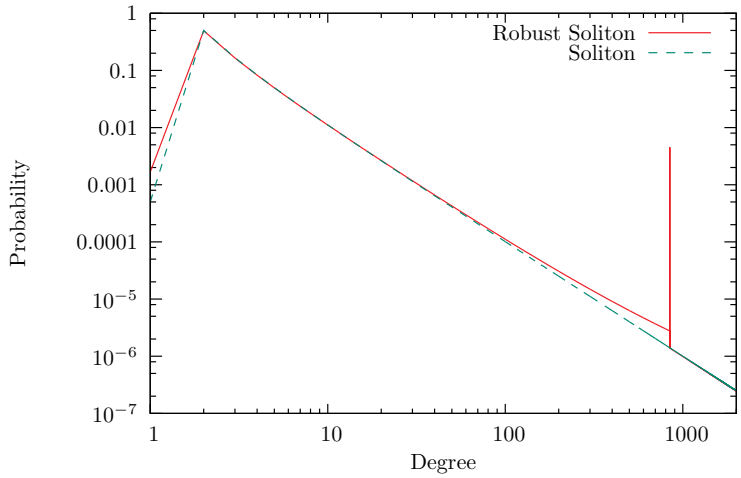
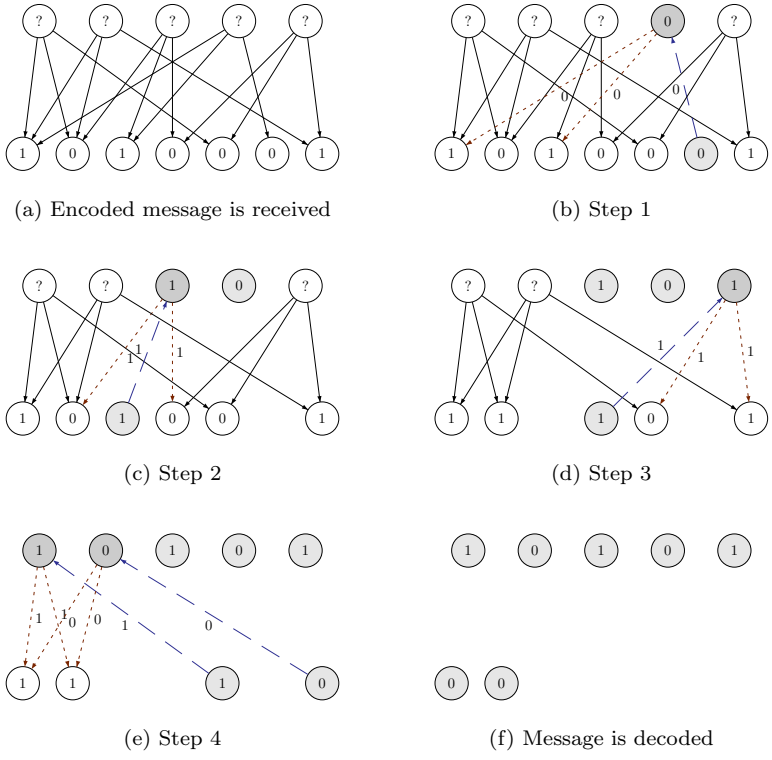


Figure 2: Distribution of degrees of output nodes for a 2000-symbols code. 49% of nodes have a degree 2 and 66% have a degree lower than 4.



(a) Encoded message is received

(b) Step 1

(c) Step 2

(d) Step 3

(e) Step 4

(f) Message is decoded

Figure 3: Belief propagation decoding. Input nodes are the original symbols and output nodes are the encoded symbols. The decoding algorithm propagates known values along edges until the whole code is decoded.

their values (i.e., XOR) is computed. The uniform choice leads to a Gaussian distribution of the number occurrences of original symbols in encoded symbols. The resulting symbol can be sent through the network.

LT codes can conveniently be represented as a Tanner Graph, which is basically a directed bipartite graph where input nodes are original symbols and output nodes are encoded symbols. Each output node is the parity bit of a set of input nodes to which it is linked. Figure 3a shows a Tanner Graph of a LT code. The first encoded symbol is $x_1 \oplus x_2 \oplus x_4 = 1$.

The encoding process implies two major properties of LT codes. These properties can easily be translated in terms of distribution of degrees in the Tanner graph representing the code. The distribution of in-degrees on output nodes must be a Robust Soliton distribution while the distribution of degrees on input nodes is a Gaussian distribution. The Robust Soliton has some interesting properties: the expected degree is $\log k$, many encoded symbols have degree 2 and there is a single spike of symbols of high degree.

Thanks to the particular properties of the resulting codes, it is possible to decode LT codes with a low complexity belief propagation decoding algorithm. The decoding algorithm works as follows: as soon as an encoded symbol has degree 1, $x_4 = 0$ for example, an original symbol can be recovered. The value is propagated along the edges of the corresponding input node (Fig. 3b). As the value of the original symbol is known, all the encoded symbols where the symbol appears can be updated by propagating the value along the edges and removing the corresponding edges. In our example, $x_1 \oplus x_2 \oplus x_4 = 0$ becomes $x_1 \oplus x_2 = 0 \oplus x_4 = 0$ (Fig. 3c). Thanks to the properties of the Robust Soliton distribution and as some encoded symbols see their degree decreased by one when recovering an original symbol (i.e., the ones that cover the recovered original symbol), there are high chances that the degree of one encoded symbol drops to one. In our example, decoding x_4 yields $x_3 = 1$, and x_3 can therefore be recovered (Fig. 3c and 3d). The value of the original symbol is in turn propagated and so on and so forth. (Fig. 3d, 3e and 3f). The decoding algorithm involves one operation per edge. Therefore, the decoding complexity is $O(k \log k)$ as the average degree is $\log k$.

RLNC have been built upon random linear codes by defining a re-encoding method. In this paper, we follow the same line of reasoning to build network codes but from another class of rateless codes. In the next section, we present our contribution, a re-encoding method for building new low complexity network codes over LT codes. We believe that extending rateless codes having a low complexity can result in new network codes offering an interesting tradeoff between efficiency and complexity. Our network codes use the LT encoder and the LT decoder described in this section. The next section presents the LT re-encoder that builds fresh encoded symbols from encoded symbols.

3 LT Network Coding

In this section, we propose a method to generate fresh encoded symbols preserving the structure of LT Codes. The structure and the efficiency of LT codes highly relies on the distribution of degrees of both input and output nodes in the Tanner graph. Even slight deviations from the Robust Soliton and Gaussian distributions strongly affect the overall performance of LT codes. Therefore, while

generating fresh encoded symbols, it is of the utmost importance to ensure that the distribution of degrees required by LT codes is respected. While this is straightforward in a centralized setting where the source stores all the original symbols, this is very challenging in a network coding scenario where intermediate nodes have to operate with only a limited number of encoded symbols. In a nutshell, our solution works as follows: when a node needs to generate a fresh symbol, it *(i)* chooses a degree d for this symbol so that the degree distribution of encoded symbols fits best the Robust Soliton, *(ii)* builds a symbol of degree d using the encoded symbols available, and *(iii)* refines the obtained symbol so that the degree distribution of original symbols matches a Gaussian distribution. The performance of each step, and thus the overall performance of LTNC, relies on efficient heuristics that fulfill the low complexity requirement. In the following, we first illustrate the rationale of LTNC on a simple example and the data structures involved. Then we dive into the details of the three steps aforementioned and the heuristics used.

As an example, from a set of encoded symbols $\{s_4, s_5 \oplus s_2, s_1 \oplus s_2 \oplus s_3, \dots\}$, the re-encoding method is able to produce a fresh encoded symbol $s_1 \oplus s_3 \oplus s_4 \oplus s_5$. The set of available symbols has two interesting properties. First, many encoded symbols are of degree 2 ($\approx 50\%$). Second, encoded symbols ($s_5 \oplus s_2$) never include a symbol of degree 1 (s_4). We exploit these properties, especially the first one, in the re-encoding process.

The available data is accessible through two data structures. First, the Tanner graph, which is used for decoding the code, allows going through relations between symbols. For example, for the original symbol s_2, it allows quickly finding combinations (i.e. encoded symbols) including such symbol ($s_2 \oplus s_5$). Moreover, for re-encoding purpose, we maintain, at decoding time, an index that maps degrees to a list of symbols of that degree. Note that this structure is used only to enable fast lookup of encoded symbols of a certain degree but it is not used for decoding.

The re-encoding method proceeds in 4 steps:

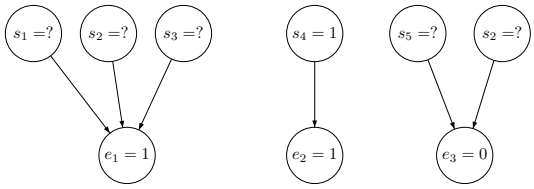
1. The re-encoding method chooses a degree d so that the distribution of degrees for the set of generated symbols remains as close as possible to the Robust Soliton distribution.

2. The re-encoding method tries to build a symbol of degree d by greedily combining available symbols of degree k with $d \geq k > 0$ starting with the highest k.

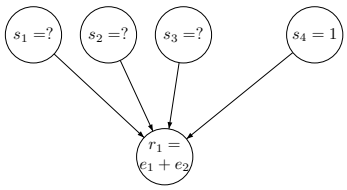
3. The candidate is refined using the supposedly large collection of symbols of degree 2. If we combine $y = s_i \oplus z$ with $s_i \oplus s_j$ and z not containing s_j, then, as $s_i \oplus s_i = 0$, we obtain $y' = s_j \oplus z$, a fresh symbol of the same degree as y. This enables to refine the distribution of degrees of input nodes to match the Gaussian distribution without jeopardizing the degree of the candidate.

4. The re-encoding method collects some statistics from past operations in an incremental manner allowing maintaining the empirical distributions. These statistics are used by subsequent re-encoding operations.

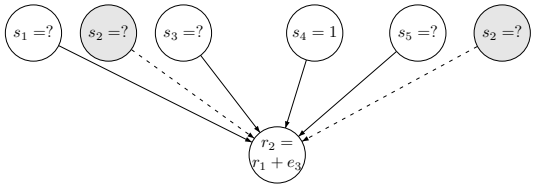
Figure 4 illustrates the two main operations of the re-encoding process. We combine encoded symbols available (Fig. 4a) to build a symbol of appropriate



(a) Our reencoder uses partially decoded data.



(b) Our reencoder builds a first symbol which degree follows the Robust Soliton distribution. We propose the degree 4 symbol $r_1 = e_1 \oplus e_2 = s_1 \oplus s_2 \oplus s_3 \oplus s_4$.



(c) Symbol s_2 has been sent too often with respect to the Gaussian distribution, it is therefore replaced by another symbol. The resulting symbol is $r_2 = r_1 \oplus e_3 = s_1 \oplus s_3 \oplus s_4 \oplus s_5$ because $s_2 \oplus s_2 = 0$.

Figure 4: LTNC re-encodes symbols to produce fresh encoded symbols

degree (Fig. 4b). Finally, we refine the distribution on input nodes using symbols of degree 2 (Fig. 4c).

We present these steps in more details in the rest of this section. We also examine how the use of a decoding method not based on Gaussian elimination impacts the whole system and propose a method to compensate such problem by detecting some of the linear dependencies using low complexity methods.

3.1 Choosing a degree

As mentioned in previous section, LT codes rely on a well-defined distribution of degrees for encoded symbols, namely the Robust Soliton. To achieve this, the empirical distribution $d \sim \tilde{A}(d)$ of degree d of encoded symbols that have been generated in the past is *(i)* maintained and updated incrementally and *(ii)* compared to the reference distribution (Robust Soliton) $d \sim RS(d)$. Then, the *deficit* of symbols $D(d) = RS(d) - \tilde{A}(d)$ is computed for each degree d and the degrees d are sorted by decreasing $D(d)$. The system then tries to generate a symbol of the degree with the highest deficit.

In order to determine if such a degree can be generated from the encoded symbols available at the node, our approach relies on a heuristic. The heuristic

needs to consider several cases. To illustrate the potential issues to face, consider the following example: from the set of symbols $\{s_i \oplus s_j, s_j \oplus s_k, s_j \oplus s_k \oplus s_l \oplus s_m \oplus s_n \oplus s_o\}$, a symbol of degree 1 cannot be generated as the smallest symbol has a degree 2. A symbol of degree 5 cannot be generated either as the sum of all degrees lower or equal to 5 is 4. Finally, a symbol of degree 4 cannot be generated as combining $(s_i \oplus s_j) \oplus (s_j \oplus s_k)$ gives $s_i \oplus s_k$, a symbol of degree 2, as the two s_js "collide". We denote this situation a *conflict*. In the next paragraphs, we derive upper bounds on the maximum degree of the encoded symbols that can be generated from available encoded blocks. This allows to discard *some* unreachable values for d.

First, a possible approach consists in using only symbols of degree $d < d'$ to build a symbol of degree d'. Assuming that no conflict occurs, the highest degree that can be generated is $\sum_{k=1}^{d} k \cdot n(k)$ where $n(k)$ is the number of symbols of degree k available.

Second, if the set of symbols $\{s_i \oplus s_j, s_k \oplus s_m, s_j \oplus s_m, s_i \oplus s_j \oplus s_k\}$ only reference 4 original symbols $\{s_i, s_j, s_k, s_m\}$, any combination of encoded symbols cannot have a degree higher than 4. Therefore, the final degree cannot be higher than the number of input nodes in the Tanner graph that are either decoded or have at least one edge.

Once a degree D has been chosen (i.e., the achievable degree corresponding to the highest deficit with respect to the Robust Soliton distribution), we start to build a symbol of such degree.

3.2 Building a symbol

This step intends at building a symbol S of a given degree D. We propose a greedy approximation algorithm that greedily adds the encoded symbols in decreasing order (with respect to their degree) thus keeping the smallest for adjusting the degree. Each symbol is used at most once.

The greedy algorithm also enforces that the degree of the symbol being generated never decreases. A recombination indeed may reduce the degree of the symbol x. For example, if $x = s_i \oplus s_j \oplus s_k$ and $y = s_i \oplus s_j$, the resulting symbol $x \oplus y = s_k$ has a degree $1 < 3$. To ensure that the quality (with respect to degree) of the candidate always increases, recombinations reducing the degree are never performed.

The algorithm for building a symbol y is given in Algorithm 1. In this algorithm, degOf(y) returns the degree of the symbol y and symbOfDeg(d) gives the set of symbols of degree d. While the degree of the symbol y being generated is lower or equals to D, the algorithm tries to combine with symbols of degree $src \leq D - \text{degOf}(s)$ that have not been used yet. The encoded symbols with the same degree are picked in a random order. When all the encoded symbols of degree src have been processed, the algorithm starts processing the set of encoded symbols of degree $src - 1$ and so on and so forth. The algorithm starts from the set of nodes of degree src and stops when either no more symbols are available or when the degree D has been reached. A recombination is performed only if it does not reduce the current degree degOf(y).

This step highly relies on the ability of the system to randomly pick a symbol of a given degree. This can be achieved efficiently using the index maintained at decoding time.

Algorithm 1 Computation of a symbol of degree D

```
1: const D                                              ▷ Objective degree
2: y ← 0                                                ▷ Symbol being built
3: used ← ∅
4: src ← D
5: while degOf(y) < D and src > 0 do
6:     srcset ← symbOfDeg(src) − used
7:     if srcset = ∅ ∨ D − degOf(y) < src then
8:         src ← src − 1
9:     else
10:        x ← pickAtRandom(srcset)
11:        used ← used ∪ {x}
12:        if degOf(y ⊕ x) ≥ degOf(y) then
13:            y ← y ⊕ x
14:        end if
15:    end if
16: end while
                                        ▷ y contains a symbol with a degree close to D.
```

3.3 Fitting input nodes degrees

This third step, namely the smoothing step, ensures that the distribution of original symbols in encoded symbols is as uniform as possible. It leverages the fact that almost half of the encoded symbols in LTNC have a degree 2 to adjust the degree distribution of input nodes. It relies on the fact that adding the encoded symbol $s_i \oplus s_j$ to an encoded symbol y that contains s_i but not s_j comes down to substituting s_i by s_j in y. Therefore, it enables to balance the degree of input nodes without jeopardizing the degree of the generated encoded symbol.

Algorithm 2 describes how the distribution of original symbols in encoded symbols is made close to uniform. In LTNC, the number of such combinations is bounded by a given threshold, further combinations tend to be useless. Our experiments showed that five combinations were sufficient. Each symbol s_i has a score $score(s_i)$. This score is proportional to the number of times it has been included in previously generated symbols and allows determining the deviation of the degree of s_i with respect to the mean degree. Symbols with a high score are over-represented in previously generated symbols. Symbols with a low score are under-represented in previously generated symbols.

The goal of the algorithm is to replace symbols with a high score by symbols with a lower (the lowest possible) score. The symbols forming y are sorted by decreasing score. The symbols are considered in order, restarting from the beginning each time a recombination is performed. For each symbol s_i, an encoded symbol $s_i \oplus s_j$ that increases the global score ($\text{score}(s_j) < \text{score}(s_i)$) is searched. If such a symbol exists, the recombination is performed. After five recombinations, the algorithm stops.

This algorithm requires a data structure that allows accessing all symbols $s_i \oplus s_j$ for a given s_i. The graph maintained at decoding time matches perfectly this requirement.

Algorithm 2 Making the distribution uniform

```
y                                          ▷ Symbol generated at step 2
AVAIL                                      ▷ Set of all available symbols

pass ← 5
comp ← listOf(x)
comp ← sort comp by decreasing score
idx ← 0
while pass > 0 do
    u ← comp[idx]
    v ← argmin_{v∈{v|(u⊕v)∈AVAILABLE}} score(v)
    if score(v) < score(u) ∧ v ∉ comp then
        y ← y ⊕ u ⊕ v
        Remove u from sorted list comp
        Add v to sorted list comp
        idx ← 0
        pass ← pass − 1
    else
        idx ← idx + 1
    end if
end while
                ▷ The final symbol y has a more uniform distribution of input nodes
```

3.4 Updating stats

Statistics on the encoded symbol generated at the node (i.e., degree distributions) are incrementally updated upon symbol generation. These statistics are used by subsequent calls to the re-encoding method. These latter are used for instance to compare the distribution of degrees of generated encoded symbols with references distributions: $\tilde{A}(d)$ is updated for each degree d and score(x) is updated for each original symbol x included in the encoded symbol generated.

3.5 Detecting redundancy

When using LT Codes, encoded symbols have a very low probability of being redundant. However, as re-encoding produces symbols over only a subspace, some non-innovative symbols might be generated. In RLNC codes, Gaussian elimination is able to detect immediately non-innovative symbols keeping only innovative ones. However, in LTNC codes, belief propagation does not include immediate detection of redundancy. In fact this is detected during the last steps of the belief propagation thus delaying redundancy detection to decoding time.

Memory and processor usages are related to the number of edges and nodes in the Tanner graph. Therefore, it is crucial to introduce a mechanism that is able to detect at reception time if an encoded symbol is innovative. If a received symbol is non-innovative it is simply discarded as it brings no additional information.

With low complexity as a first objective, we propose a mechanism for detecting non-innovative symbol consisting of two checks. First, it checks if an identical symbol has already been received. Second, it checks, for symbols of degree 2, if the received symbol is linearly dependant with any set of received symbols of degree 2.

The first mechanism builds a unique signature $u(x)$ for each received encoded symbol x and searches or inserts the signature in a data-structure allowing fast search and insertion (such as a binary tree or an hash set). In order to limit the complexity and since most of the symbols have a small degree, we limit the mechanism to symbols of degree lower or equal to 3. The function u is a low complexity function that produces an identifier. The unique signature corresponds to an integer formed by the concatenation of the sorted ids of original symbols composing the encoded symbol x. Therefore, when a symbol x of degree 1, 2 or 3 is received, we search $u(x)$ in the data-structure. If it is found, the message is non-innovative and the message is deleted. Otherwise, the message is accepted for decoding and $u(x)$ is added to the data-structure.

The second mechanism aims at detecting redundancy between symbols of degree 2. It is important to notice that removing a non-innovative symbol of degree 2 does not have any impact on the decodability using belief propagation decoding. To this end, we design an online algorithm that handles symbols when they are received. An encoded symbol of degree 2 is non-innovative if it can be generated from the set of encoded symbols available. Considering the graph which vertices are the original symbols and there is an edge between node a and node b if the encoded symbol $a \oplus b$ is available, determining if an encoded symbol of degree 2, say $s = a \oplus b$ is non-innovative comes down to check whether adding the edge (a, b) creates a cycle in the graph. As an example, consider the following set of symbols received $\{a \oplus b, b \oplus c, c \oplus d, d \oplus b\}$, $d \oplus b$ is non-innovative as it creates the cycle (b, c, d). In other words, one must check whether a and b lie in the same connected component. Based on this remark, we design a detection technique that designate a leader for each connected component and stores for each node s of the graph, the leader of the connected component it belongs to denoted $L(s)$. Doing this, it can easily be detected whether the encoded symbol received is innovative or not. Initially, $L(s)$ is set to s for all s. On one side, when an innovative edge (a, b) is added to the graph, the connected components of its two ends a and b are merged. Assuming a's connected component is smaller than b's, this is achieved by setting $L(s)$ to $L(b)$ for all s such that $L(c) = L(a)$. On the other side, a non-innovative edge $(a, b)$is detected when the leaders of the two ends are the same $L(a) = L(b)$.

The method for detecting redundancy has a low complexity. The first method proposed implies looking up in a binary tree with n nodes. Therefore, the complexity is $O(\log n)$. The second method also has a complexity of $O(\log n)$. Indeed, the worst case happens when merges performed updates many values of $L(x)$. As the small component is always merged in the big components, the worst case is when components have the same size. Let us consider the binary tree representing how connected components are merged two by two. The tree is balanced and complete as merged components have the same size. It has n nodes has n merges are performed until all n nodes are in the unique final connected component. Therefore, the height of the tree is $\log_2 n$. At level i $(i = 0 for leaves)$, $n/2^{i+1}$ connected components of size 2^i are merged involving $2^i \cdot n/2^{i+1} = n/2$ updates. Hence, at each level in the tree $n/2$ nodes gets an updated leader: the total number of updates of leaders, for n received symbols, is $O(n \log n)$. The complexity per received symbol is, therefore, $O(\log n)$.

4 Performance

We evaluate LTNC and compared it against RLNC and a scheme without codes (i.e., where nodes forward only initial symbols chosen uniformly at random) using a cycle-based simulator. For the sake of fairness, especialy when dealing with CPU cycles, the same set of optimizations at compilation time are used for the three solutions.

We consider a peer-to-peer network of size N, where each peer is connected to a static set of d other peers chosen uniformly at random. Each cycle, a peer can transmit one symbol. All peers run the same algorithm except one, the source, that produces and emits the L symbols to be broadcast.

Note that even though the communications are emulated, the coding and decoding parts are effectively executed when running the simulation. Therefore, simulating such an application requires a significant amount of time, especially for RLNC. This forces the number of peers to be limited to 200 to account for a reasonnable amount of simulation time. Results presented have been averaged over 10 Monte-Carlo experiments.

We evaluate the performance of LTNC against two references schemes that captures the trade-off between the performance with respect to content-dissemination (i.e., speed of convergence and message complexity) and the computationnal complexity of operations performed at the nodes (i.e., re-encoding, decoding, ...).

RLNC: At the beginning of each cycle, every peer chooses uniformly at random one of its neighbors to upload data to and sends a random linear combination of known symbols using RLNC. In our setting, RLNC operations are performed over $GF(2^8)$ with a sparseness coefficient of 20. The sparseness coefficient allows lowering the cost of encoding and re-encoding operations while preserving global performance. This set of parameters is widely acknowledged as the optimal settings for linear network coding.

Without Codes: This scheme does not use any form of coding (even at the source) but only a push-based epidemic dissemination in which a peer receiving an innovative symbol (i.e., an original symbol it has not received yet) forwards the symbol to f other peers. On average, $O(N \log N)$ messages transmissions are required to reach all N peers. This is an upper bound for our evaluation of LTNC. Our proposition must outperform this scheme.

In the following, we give details on the metrics used for evaluation.

4.1 Time to complete

We measure the time needed so that 95% of peers in the network receive enough encoded symbols to successfully recover the original symbols. Figure 5 shows such time for several values of code lengths and several network sizes. Surfaces show the results for each code. As expected, LTNC obviously outperforms w/o codes scheme while exhibiting a reasonable time overhead (33%) compared to RLNC.

The superiority of LTNC increases with the code length. The results for the longest codes considered within the biggest network are shown on Figure 6b.

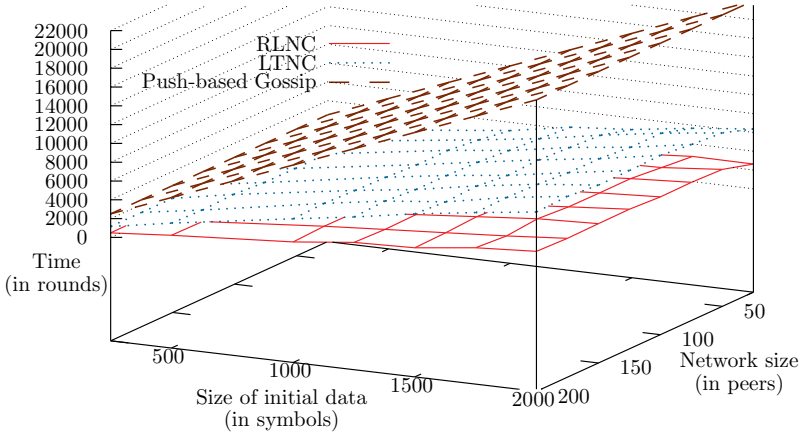


Figure 5: Time needed for 95% of peers to complete, for various code lengths and network sizes. LTNC outperform the policy without codes and perform quite well when compared to RLNC.

Time and number of messages needed to complete the dissemination are given in Table 1.

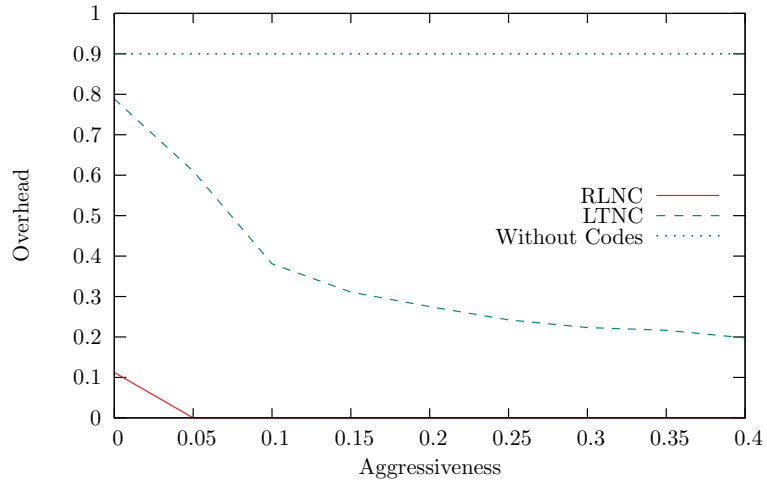
4.2 Impact of aggressiveness

The aggressiveness is a parameter that defines when a peer starts to generate and send re-encoded symbols. An aggressiveness of 20% means that a peer starts sending data if it has received 20% of the data needed to recover original symbols with high probability. The lower the value the more aggressive are the peers. Increasing the aggressiveness value is known to decrease overhead and enhance network coding performance at the cost of a slower startup. We study the impact of aggressiveness on LTNC and RLNC codes. For this evaluation, the code length has been set to 2000 symbols and the size of the network to 200 nodes.

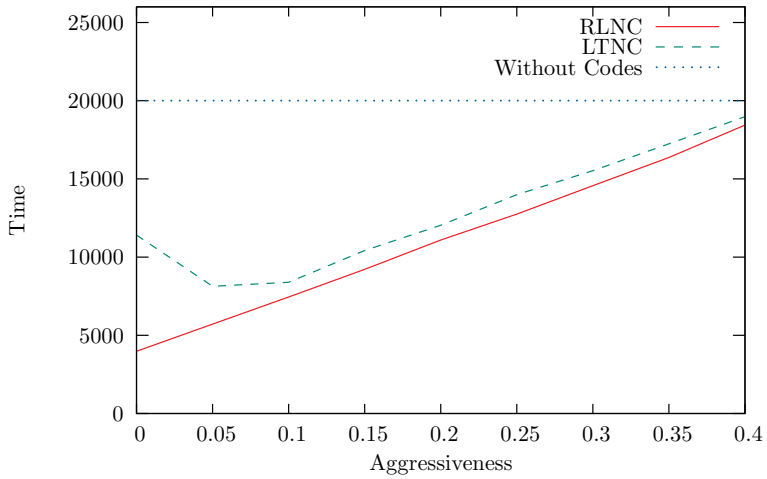
We plot the time needed to complete and the overhead. These results are, as previously, given for 95% of peers to complete. The overhead is defined as the fraction of non-innovative (redundant) symbols sent over the total traffic. Hence, an overhead of 90% means that out of 20,000 total messages, 18,000 of them were useless and only 2,000 were useful.

As shown on Figure 6a, using LTNC with a highly aggressive behaviour, where nodes start to send data at the early stages of the dissemination, increases the redundancy of data circulating in the network. This same behavior is observed for RLNC codes. However, contrary to RLNC, starting to re-encode early also reduces the performance of LTNC as shown on Figure 6b.

Intuitively, the time to complete should increase with the aggressiveness value (peers wait more before cooperating). However, for LTNC, it enhances the overall performance. From this, we can conclude that being extremely aggressive (value set to 0.0) makes the peers to send redundant symbols therefore introducing more redundant data, which in turn perturbs subsequent recombi-



(a) Overhead



(b) Time to complete

Figure 6: Impact of aggressiveness when 95% of the peers are complete. The system contains 200 peers and use code of length 2000.

nations; hence, reducing the aggressiveness has a positive impact on the performance of LTNC. When using LTNC, peers must not be too aggressive at re-encoding. An aggressiveness coefficient of 0.05-0.10 seems reasonable.

Figure 6b and Figure 6a compare LTNC to reference schemes. Table 1 summarizes all results. RLNC are shown in two settings, one that is optimal with respect to time needed to complete and one that is optimal with respect to the overhead. LTNC and Without codes are shown at their optimal setting which is both optimal with respect to time and with respect to overhead.

4.3 Impact of fitting input nodes degrees

We have evaluated the effect of the step fitting input nodes degrees, namely the smoothing step. We simulated the dissemination process of a code of length 2000

	Time	Overhead	Messages
RLNC	**4800**	10.0 %	2222
	6000	**0.0 %**	2000
LTNC	**8000**	**38.5 %**	3252
Without codes	**20000**	**90.0 %**	20000

Table 1: Optimal performances for 95% of peers to complete in a network of 250 peers and a code of length 2000.

over 200 nodes with and without the normalizing step, with an aggressiveness of 0.10.

The time to complete is 8000 for 38.5% of overhead with the smoothing step and 11500 for 58.9% of overhead without the smoothing step. On Figure 7, we plot the distribution of degrees for input nodes. Clearly, the smoothing step makes the distribution on input nodes more uniform resulting in a narrower distribution of degrees. As a consequence, the system is more efficient.

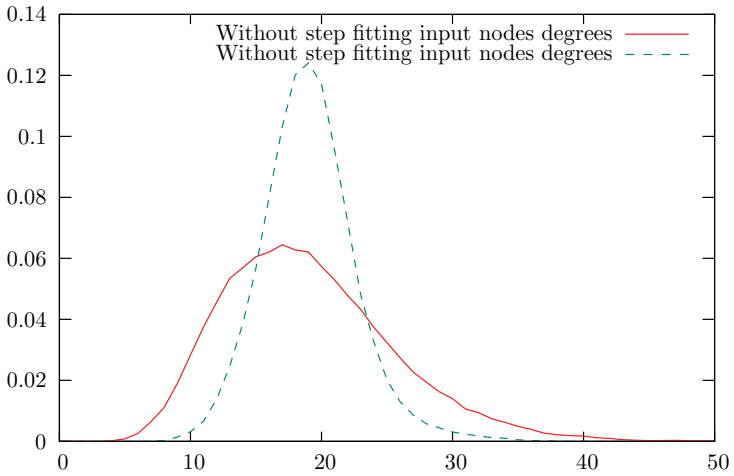
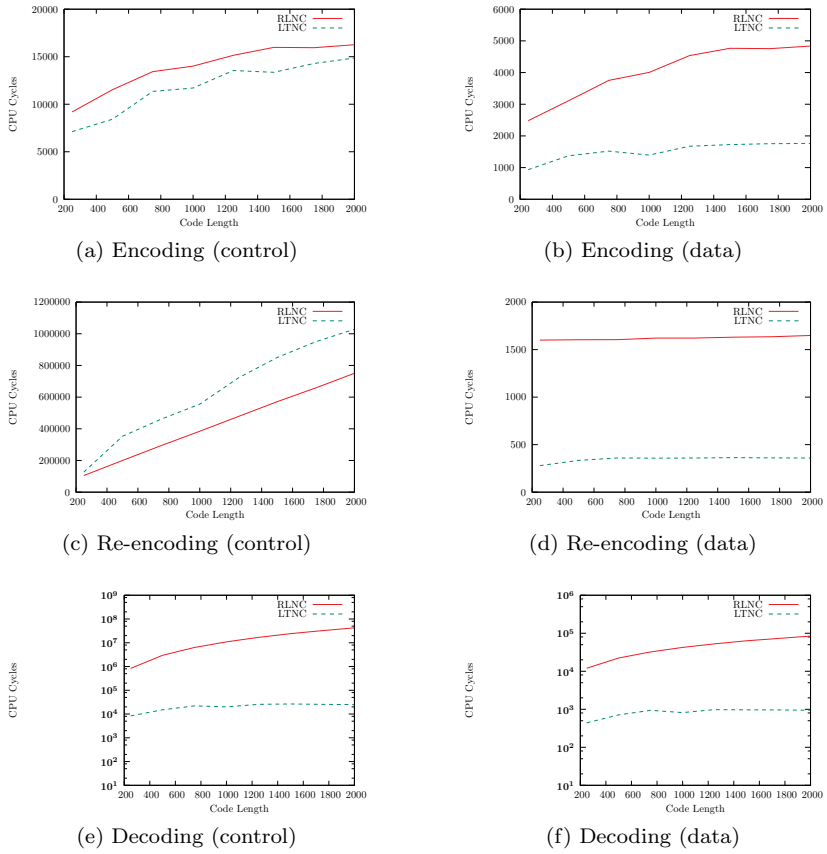


Figure 7: Distribution of degrees on input nodes.

4.4 Computational costs

LTNC exploits a decoding method with a lower complexity than the one used in RLNC. In this sub-section, we are interested in comparing the cost of operations in LTNC with the cost of operation in RLNC.

We evaluated analytically the complexity of the operation in LTNC codes. The results are summarized in Table 4.4. All complexities are shown "per symbol". Therefore, for transmitting n symbols, if $O(n^2)$ operations are needed for decoding, it means that the global cost of decoding is $O(n^3)$. *Receiving* involves detecting redundancy through Gaussian elimination or proposed algorithms. *Decoding* corresponds to the cost of the final decoding step. The whole cost of decoding is shown as *Total decoding*, it corresponds to *Receiving* and *Decoding*. The re-encoding operation of LTNC involves four steps. The first step contains the more complex control processing as it sorts degrees ($O(n \log n)$)



(a) Encoding (control)

(b) Encoding (data)

(c) Re-encoding (control)

(d) Re-encoding (data)

(e) Decoding (control)

(f) Decoding (data)

Figure 8: CPU cycles for each operation. We show both results for the control and data operations The graph shows that we strongly reduce the cost of the decoding operation by slightly increasing control cost for the re-encoding operation.

by their deficit. The worst case for processing data is when the second steps builds a symbol of size $\log n$ on average by merging symbols of size 1. As, at most, a constant number of recombinations are performed in the third step, the complexity is $O(m \log n)$.

We also measure the number of CPU cycles per symbol encoded, re-encoded or decoded. Processing each symbol implies *control* processing that includes inverting the matrix, handling the graph used for decoding or computing the coefficient used. It also implies *data* processing that includes computing the useful data. In our simulations, we use small blocks (4 bytes), therefore, *control* processing strongly dominate our results. However, if block size is increased, the number of cycles devoted to *data* processing grows linearly with block size. Therefore, *data* processing dominates for large block size.

We present our results on Figure 8. For the sake of readability, plots for decoding operations (8e and 8f) are shown with a logarithmic scale while other plots have a linear scale. With respect to CPU cycles used for encoding/re-encoding operations, LTNC and RLNC have similar performances (8a and 8c) with a slight advantage to LTNC codes when processing data (8b and 8d).

	w/o codes	RLNC	LTNC
Encoding	$O(1)$	$O(k)$	$O(\log n)$
Re-encoding	$O(1)$	$O(k)$	$O(n \log n)$
Receiving	$O(1)$	$O(n^2)$	$O(\log n)$
Decoding	$O(1)$	$O(n)$	$O(\log n)$
Total Decoding	$O(1)$	$O(n^2)$	$O(\log n)$

(a) Complexity for processing control (matrix inversion, graph decoding) (per symbol)

	w/o codes	RLNC	LTNC
Encoding	$O(m)$	$O(km)$	$O(m \log n)$
Re-encoding	$O(m)$	$O(km)$	$O(m \log n)$
Receiving	$O(m)$	$O(nm)$	$O(m)$
Decoding	$O(m)$	$O(nm)$	$O(m \log n)$
Total Decoding	$O(1)$	$O(mn)$	$O(m \log n)$

(b) Complexity for processing data (per symbol)

	w/o codes	RLNC	LTNC
Control	$O(n)$	$O(n^2)$	$O(n \log n)$
Data	$O(nm)$	$O(nm)$	$O(nm)$

(c) Space Complexity (global, for n symbols)

Table 2: Complexity of operations for LTNC and references (RLNC and Without (w/o) codes). Note that k may be $c \cdot \log n < k < n$. All complexities are shown for control structure and for data. The complexity for data will dominate the complexity for structure if block size is big enough.

However, when considering the decoding operation (shown in logarithmic scale), LTNC clearly outperform RLNC for processing both control (8e) and data (8f). Moreover, the advantage of LTNC over RLNC increases as the code length increases.

4.5 Efficiency of redundancy detection

Next, the effectiveness of two redundancy detectors is evaluated. The first, a key-based detector, detects identical symbol using fast lookup in a binary tree. The second, a cycle-based detector, detects linearly dependant symbols of degree 2. As they both detect identical symbols of degree 2, their results are depicted individually and combined. We present the number of total useless symbols (the optimal that we could get) and the amount of useless symbols that have been detected without decoding.

Figure 9 shows the result for various values of aggressiveness in a 200 peers system using codes of length 2000. Clearly, the redundancy detectors are efficient as they detect almost half of the redundant symbols without decoding (using a very low complexity procedure).

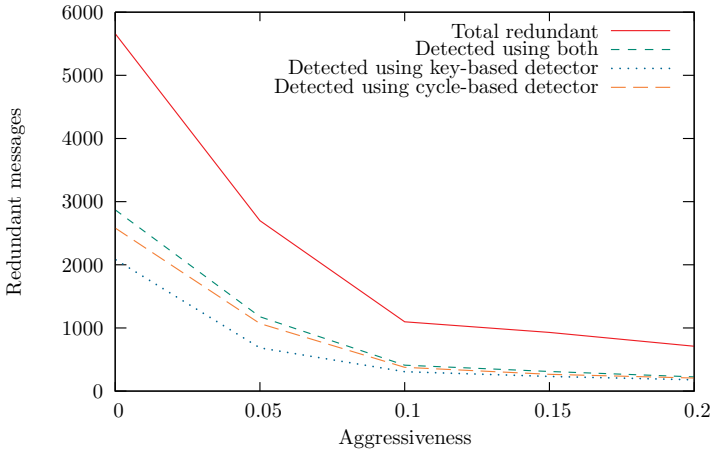


Figure 9: Redundancy detectors allow detecting almost half of the redundancy at reception time. They allow to save some memory and lower decoding costs that are slightly increased by the redundancy.

5 Related Work

In this section, we review work related to RLNC and LTNC and clearly position LTNC with respect to them. Table 3 gives an overview of the taxonomy of network codes together with the underlying rateless codes.

Motivated by the high decoding cost of RLNC, a large amount of research has been devoted to designing low complexity network codes. A noticeable piece of work is the Raptor-based networks codes proposed in [18]. Similarly to linear network coding, the proposed solution generates encoded symbols by XORing the encoded symbols received. However, Raptor codes [16] relies on a carefully chosen distribution of degrees of encoded symbols as LT codes do. In their re-encoding method, the author simply mix encoded symbols together without caring about the degree of the resulting symbol. Degrees tend to increase with the number of recombinations thus breaking the properties ensuring low complexity decoding of Raptor codes. Hence, Raptor network video coding decoding still relies on the very same Gaussian elimination as RLNC.

Growth codes [10] are parity codes, using belief propagation as the decoding algorithm. These codes were built to collect data in sensor networks. Therefore, each sensor has one piece of data and all sensors should get all pieces of data. Each sensor periodically exchanges random symbols of data with its neighbors. In order to avoid the Coupon Collector problem, time is divided into slots and the degree of sent symbols increases with the slot number. During the ith slot, symbols of degree i are sent. This way, the probability of getting a redundant symbol is kept constant. Note that this defines more a *Distributed Channel Coding* method than a *Network coding* method. While this proposal is interesting and presents many similarities with network codes, it tackles a different problem. Initially, each node has a subset of the data and they all start exchanging information at the same instant. Moreover, all the nodes in the network must be loosely synchronized so that they progress at the same speed and use appropriate slice.

Distributed source coding is the third most relevant scheme. Although it is less powerful than network coding, it deserves some attention. Distributed source coding allows to balance the load of encoding data over multiple processors. In distributed source coding, the mixing operation must be applied once and only once, otherwise, the properties of the code are lost. In [15], the authors define Distributed LT Source codes, building a distribution that can use a simple XOR as the mixing operation. The resulting codes are regular LT codes where degree of output nodes follows the Robust Soliton Distribution. Therefore, the load of encoding is distributed on multiple processors. However, this does not define a network code in which a node can re-encode before having decoded.

Rateless codes, Distributed source codes and Network codes are related as summarized on Figure 10. Network codes define a new re-encoding operation. Distributed codes [15] define a mixing operation. Network codes and distributed source codes are built upon Rateless codes by adding an operation. In network codes, the result of a re-encoding operation can be used as the input of a re-encoding operation whereas in distributed source codes, the result of the mixing operation cannot be used as the input of the mixing operation. In fact, the code acquires its structure after the mixing operation has been applied once and the mixing operation must not be applied more than once.

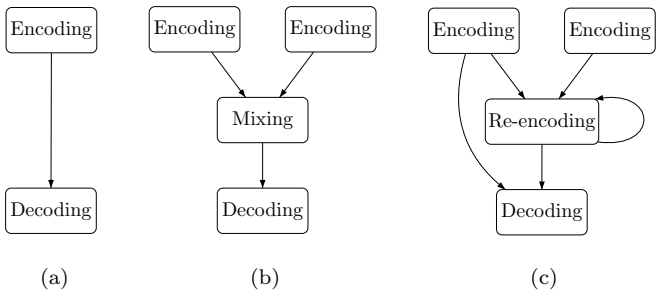


Figure 10: Rateless Codes (a), Distributed Source Codes (b) and Network Codes (c) are related.

Finally, by allowing internal nodes of a network to regenerate new encoded symbols, network coding breaks existing signature schemes. Nodes could re-encode symbols introducing errors and forward those erroneous symbols. Each node would, therefore, include this corrupted symbol in every re-encoded symbols it sends. Network coding without signature scheme allows a malicious node to pollute every symbol in the network. To address this issue, signature schemes for linear network codes have been proposed [20, 2, 7, 9]. As we perform only linear ($\oplus$) recombinations of linear symbols, our codes remain linear. Therefore, any of the signature schemes sub-cited can be used to provide security when using LTNC.

6 Conclusion

In this paper, we propose a new low complexity network code. Current codes mostly rely on random linear network codes that require a computationally

Inspiring Rateless Code	Network Code	Re-encoding	Decoding
Random Linear Codes	Random Linear Network Codes and variants	Compute a Random Linear Combination	High Complexity, Gauss Elimination
LT Codes and Raptor Codes	Raptor Video Network Coding	XOR multiple symbols together	
	Growth Codes	Distributed channel, encoding degree increase each slice of time	Low complexity, Belief propagation
	LT network codes	Re-encoding method proposed in this paper	

Table 3: A taxonomy of network codes

intensive Gaussian elimination to be decoded. Instead, we propose to extend rateless LT codes to turn them in low complexity network codes (LTNC).

In this paper, we propose a re-encoding method allowing nodes that have received data but not yet decoded it to re-encode such data and produce new redundancy blocks. Mechanisms to detect redundant messages as soon as possible have also been designed (as belief propagation, contrary to Gaussian elimination does not detect linearly dependant symbols immediately). The resuting network codes exhibit a much lower decoding complexity ($O(\ln n)$) than random linear codes ($O(n^2)$) per symbol decoded for a code of length n. These network codes offer a nice tradeoff between coding efficiency (communication) and decoding cost (complexity).

In this paper, we have successfully built network codes with a low complexity by building upon existing rateless codes as demonstrated by the evaluation. As this approach turns out successful, it may inspire future works toward low complexity network codes based on other extensions of fountain codes.

References

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung. Network information flow. *IEEE Transactions On Information Theory*, 46(4):1204–1216, July 2000.

[2] D. Charles, K. Jian, and K. Lauter. Signature for network coding. Technical Report MSR-TR-2005-159, Microsoft, 2005.

[3] P. Chou and Y. Wu. Network coding for the internet and wireless networks. *IEEE Signal Processing Magazine*, 24(5):77–85, September 2007.

[4] B. Cohen. Incentives build robustness in bittorrent. In *P2PEcon*, June 2003.

[5] C. Gkantsidis, J. Miller, and P. Rodriguez. Anatomy of a p2p content distribution system with network coding. In *IPTPS*, 2006.

[6] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *INFOCOM*, 2005.

[7] T. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, M. Effros, and D. R. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *ISIT*, 2003.

[8] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transaction on Information Theory*, 52, October 2006.

[9] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Médard. Resilient network coding in the presence of byzantine adversaries. In *INFOCOMM*, 2007.

[10] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth codes: maximizing sensor network data persistence. In *SIGCOMM*, 2006.

[11] M. Luby. LT Codes. In *FOCS*, 2002.

[12] G. Ma, Y. Xu, M. Lin, and Y. Xuan. A content distribution system based on sparse network coding. In *NetCod*, 2007.

[13] D. J. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2002.

[14] M. Mitzenmacher. Digital fountains: A survey and look forward. In *ITW*, 2004.

[15] S. Puducheri, J. Kliewer, and T. E. Fuja. Distributed LT Codes. In *ISIT*, July 2006.

[16] A. Shokrollahi. Raptor codes. *IEEE/ACM Transactions on Networking*, 14:2551–2567, June 2006.

[17] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27:533–547, 1981.

[18] N. Thomos and P. Frossard. Raptor network video coding. In *MV*, September 2007.

[19] M. Wang and B. Li. How practical is network coding? In *IWQoS*, 2006.

[20] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. In *INFOCOMM*, 2008.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

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