LT Network Codes

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

LT Network Codes

Mary-Luc Champel*, Kévin Huguenin†, Anne-Marie Kermarrec‡ and Nicolas Le Scouarnec*
*Technicolor, Rennes, France
†IRISA / Université de Rennes 1, Rennes, France
‡INRIA Rennes - Bretagne Atlantique, Rennes, France

***Abstract*—Network coding has been successfully applied in large-scale content dissemination systems. While network codes provide optimal throughput, its current forms suffer from a high decoding complexity. This is an issue when applied to systems composed of nodes with low processing capabilities, such as sensor networks.**

In this paper, we propose a novel network coding approach based on LT codes, initially introduced in the context of erasure coding. Our coding scheme, called LTNC, fully benefits from the low complexity of belief propagation decoding. Yet, such decoding schemes are extremely sensitive to statistical properties of the code. Maintaining such properties in a fully decentralized way with only a subset of encoded data is challenging. This is precisely what the recoding algorithms of LTNC achieve.

We evaluate LTNC against random linear network codes in an epidemic content-dissemination application. Results show that LTNC increases communication overhead (20%) and convergence time (30%) but greatly reduces the decoding complexity (99%) when compared to random linear network codes. In addition, LTNC consistently outperforms dissemination protocols without codes, thus preserving the benefit of coding.

I. Introduction

Over the last decade, the decentralized multicast approaches targeting large-scale systems have been extensively studied, yielding efficient dissemination schemes such as epidemic protocols. In this context, network coding, initially proposed by Ahlswede *et al.* [1], has proven to be a powerful paradigm significantly improving the throughput. This has been successfully applied both in wired systems (e.g., p2p file sharing with Avalanche [2], [3]), wireless systems (e.g., sensor networks with [4], [5] and mesh networks with [6]). While unencoded epidemic approaches provide robustness in the dissemination by implementing a lot of redundancy, network coding techniques introduce a smarter redundancy scheme, providing at a lower cost an optimal solution with respect to dissemination. Such schemes rely on a recoding procedure achieved at each step of the dissemination chain, where nodes involved in the dissemination recode content in the form of linear combination of received packets. Despite the success of network coding, some works have shown that one of the limitations in current forms of network coding, namely random linear network codes (RLNC), is that they require a high complexity decoding process. Some optimizations [7], [8], [9] have been proposed. Yet, none removes entirely the complexity of the decoding method, namely $\mathcal{O}\left(m \cdot k^2\right)$ of Gauss reduction in RLNC (typical network codes) where the content disseminated is split in k native packets of size m.

In this paper, we propose LTNC, a novel approach to build network codes from low complexity rateless erasure codes, namely LT codes [10], alleviating high complexity decoding procedure at the nodes. This is of the utmost importance when nodes have limited computational power typically in sensor networks composed of low capability nodes. LT codes enable a low-complexity decoding thanks to the belief propagation decoding scheme, which recovers native packets in $\mathcal{O}\left(m \cdot k \log k\right)$. Yet, such schemes are extremely sensitive to specific statistical properties of encoded packets. Such properties are challenging to implement in fully decentralized settings where nodes have only *some encoded* packets available when recoding. To the best of our knowledge, LTNC is the first network coding technique based on LT codes, thus enabling the use of belief propagation for decoding. There has been other attempts to distribute encoding of LT Codes or propose distributed encoding scheme using belief propagation for decoding method [11], [4], [12], [5]. However, other attempts build encoded packets only by combining native ones, limiting the range of applications that could benefit from such schemes. Instead, our scheme enables to recode fresh encoded packets at each node, from encoded packets while preserving the statistical properties of LT codes. With LTNC, the freshly recoded packets preserve the structure and properties of LT codes to maintain decodability using the low complexity decoding algorithm. Since LTNC are linear network codes, traditional optimizations (e.g., generations [2], [13]) and security schemes (e.g., homomorphic hashes and signatures [14], [15], [16], [17]) can be directly applied. This enables the use of LTNC in practical content dissemination systems such as Avalanche [3].

We evaluate LTNC in an epidemic content-dissemination application. Note that beyond content dissemination applications, LTNC can be applied to self-healing distributed storage as the recoding method can be used to build new LT-encoded backups in a decentralized fashion similarly to [18], [19] that use traditional random linear network codes. Our experiments show that, for a code length of $2,048$, LTNC reduces the computational complexity of decoding by 99% at

the price of a communication overhead of 20%. In addition, LTNC consistently outperforms unencoded dissemination protocols, thus preserving the benefit of coding.

The rest of this paper is organized as follows. Section II gives some background about network coding and LT codes. Section III provides a high level overview of LTNC and describes in detail the algorithms involved in the recoding method. An evaluation of LTNC is presented in Section IV. Section V reviews the related work and Section VI concludes the paper.

II. Background

Coding techniques have been widely and successfully used in push-based dissemination applications where content, divided into k *native packets* $\{x_i\}_{i=1}^{k}$ of size m, is broadcast from one or multiple sources to a set of nodes connected by a network. With erasure coding, the k native packets are combined *at the source* into $n > k$ *encoded packets*, and can be *recovered at the nodes* from any set of $(1+\varepsilon)\cdot k$ encoded packets ($\varepsilon \geq 0$). Intermediary nodes of the network taking part in the dissemination simply forward encoded packets to their neighbors.

An example of such codes are linear codes which generate as encoded packets linear combinations, over a Galois field, of native packets. Encoded packets are sent over the network with their associated code vectors of size k that describe the coefficients of the linear combinations. For instance, for $k = 4$, the code vector of the encoded packet $x_1 \oplus x_3$ is $(1, 0, 1, 0)$. Upon reception, an encoded packet is stored in memory and its code vector is usually appended (as a row) to the so-called *code matrix*. As soon as the code matrix is full-ranked, native packets are recovered using Gaussian reduction in $\mathcal{O}\left(m \cdot k^2\right)$ operations. Linear codes are simple as they consist only in xor operations. In addition, they are *rateless* [20] since the number of distinct encoded packets they can generate grows exponentially with the number of native packets (2^k here) resulting in close to optimal decoding performance ($\varepsilon \approx 1$).

With random linear codes [21], where the coefficients of the linear combinations are chosen uniformly at random, optimal coding is achieved without the need for any coordination between nodes: native packets can be recovered from k encoded packets with high probability using Gaussian elimination.

In [10], Luby proposed Luby Transform codes (LT), a low complexity approach to linear codes. Similarly to linear codes (which they inherit), LT codes involve linear combinations of native packets. Linear combinations are randomized and performed over GF(2). However they differ from random linear codes in that *(i)* they specify statistical properties on the encoded packets sent and *(ii)* they are decoded using a low complexity algorithm called *belief propagation* (that uses a dedicated data structure instead of a code matrix). Belief propagation requires only $\mathcal{O}\left(mk \cdot \log k\right)$ operations but relies on a specific distributions of native and encoded packets. LT encoded packets are organized into a specific data structure named a *Tanner graph* [22]. A Tanner graph is a bipartite graph where nodes in the first set are native packets and the nodes in the second set are the encoded packets received. There exists an edge from a native packet x to an encoded packet y if x is involved in the linear combination forming y. The degree of a packet is the number of edges originating from (resp. pointing to) this particular node and is denoted by $d(\cdot)$. Figure 1 depicts an example of a Tanner graph. Every time a native packet x is received (i.e., an encoded packet of degree 1) or decoded, every encoded packet y involving x (i.e., to which x points) is xor-ed with x and the edge between x and y is deleted. When a native packet is the only one to point to an encoded packet, it can be decoded and its value is propagated along its outgoing edges (each encoded packet to which it points is xor-ed with the decoded native packet).

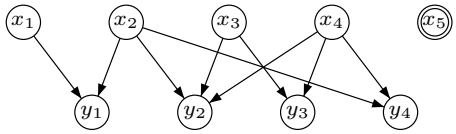


Figure 1. An example of a Tanner graph: $y_4 = x_2 \oplus x_4$, x_3 has degree 2 and y_2 has degree 3. x_5 has been decoded.

It is clear from the previous paragraph that the belief propagation decoding algorithm requires at least one encoded packet of degree one. More generally, the lower the degree of the encoded packets the faster the decoding. On the other hand, the higher the degree of the encoded packets, the less redundant the sent packets. It is shown in [10] that the optimal distribution of degrees for the encoded packets is the Robust Soliton (RS), depicted in Figure 2. The RS distribution is composed of more than 50% of encoded packets of degree 1 or 2 allowing to bootstrap belief propagation, and an average degree of $\log k$ resulting in low complexity decoding. Secondly, to ensure optimal decoding, all native packets must have roughly the same degree. In other words, the distribution of degrees of the native packets must have a minimum variance (ideally a Dirac).

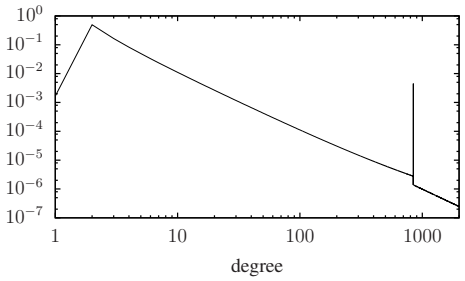


Figure 2. Robust Soliton: optimal distribution of degrees for encoded packets.

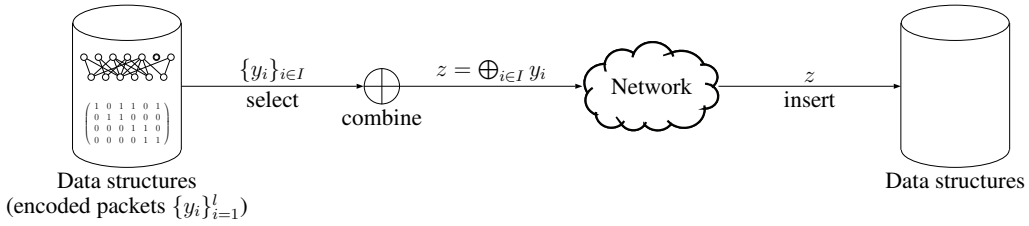


Figure 3. Global picture of (Linear) Network Coding.

Yet efficient, erasure codes are suboptimal since only the source can increase the packets diversity by generating distinct encoded packets (i.e., intermediary nodes only forward the packets they receive). In network coding [1], intermediary nodes are able to generate *fresh* encoded packets from the encoded packets they received, namely *recoding*, as illustrated in Figure 3. This results in a higher diversity of the encoded packets circulating in the network leading to increased performance as compared to erasure coding: network coding allows the dissemination throughput to reach the network capacity. Linear codes are well suited for network coding as linearly combining encoded packets results in fresh encoded packets. Random linear codes for instance can easily be turned into random linear network codes (RLNC) by recoding encoded packets received into fresh ones using random linear combinations. In other words, the recoding operation is the same as the coding operation except that it operates on encoded packets instead of native packets. However, building network codes from LT codes requires intermediary nodes to be able to generate, with only partial information, encoded packets which degrees follow a specific distribution while keeping the variance of degrees of native packets low. Effectively, while this can easily be achieved at the source where all native packets are available, this is very challenging when a node has only some encoded packets available. In the following, we describe LTNC, network codes based on Luby Transform.

III. LT Network Coding

In this section we present LTNC, low complexity network codes based on Luby Transform for push-based content dissemination applications where nodes periodically send possibly encoded packets to their neighbors. As mentioned in the previous section, low complexity decoding can be obtained using the belief propagation algorithm which efficiency highly relies on statistical properties of encoded packets available at the node. More specifically, the degree distributions of native and encoded packets must respectively match a Robust Soliton and a Dirac. Therefore, it is of the utmost importance to ensure, when recoding encoded packet into fresh ones, that the structure of LT codes is preserved. This problem is especially challenging in a network coding scenario where intermediary nodes operate with a limited number of encoded packets available.

In a nutshell, our solution works as follows: when a node needs to generate a fresh encoded packet (i.e., recode), it *(i)* builds a packet of degree d, where d is drawn from a Robust Soliton distribution, using the encoded packets available; *(ii)* refines the obtained packet so that the variance of the distribution of degrees of native packets is reduced. The first step involves NP-Complete sub-problems and thus cannot be solved at a low computational cost. The performance of each step of the recoding method, and thus the overall performance of LTNC relies on efficient heuristics and complementary data structures allowing low complexity recoding with a good approximation of the structure of LT codes.

In the following, we first give the rationale behind LTNC and illustrate its functioning and the data structures used on a concrete example. Table I (page 4) summarizes the different data structures used by LTNC with their purpose. We then dive into the algorithmic details of the two aforementioned steps. Finally, we present various optimizations for LTNC including application-specific optimizations that rely on some features available on the application framework (e.g., feedback channel for Internet applications).

A. Overview of LTNC

Consider the example depicted in Figure 4, where a node p recodes a fresh encoded packet from previously received ones. The initial content is split into $k = 7$ native packets and node p stores 6 encoded packets $\{y_i\}_{i=1}^{6}$ and the native packet x_6.

First, p picks a random degree d drawn from the Robust Soliton distribution for the packet to be recoded (step 1.1). The degree distribution is an input of the system, fixed in advance to its optimal value (i.e., Robust Soliton [10]) and known at each node. In the example of Figure 4, the picked value is $d = 5$.

The node p tries to build, by linearly combining previously received encoded packets and decoded native packets, a fresh encoded packet of degree $d = 5$ (step 1.2). To this end, p maintains an index that maps degrees to a list of encoded packets of each particular degree allowing fast lookup of encoded packets of a given degree. In the example of Figure 4, p picks two encoded packets y_1 and y_2 of respective degrees 2 and 3 and builds a fresh encoded packet $z = y_1 \oplus y_2$. In terms of native packets, $z = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$ and

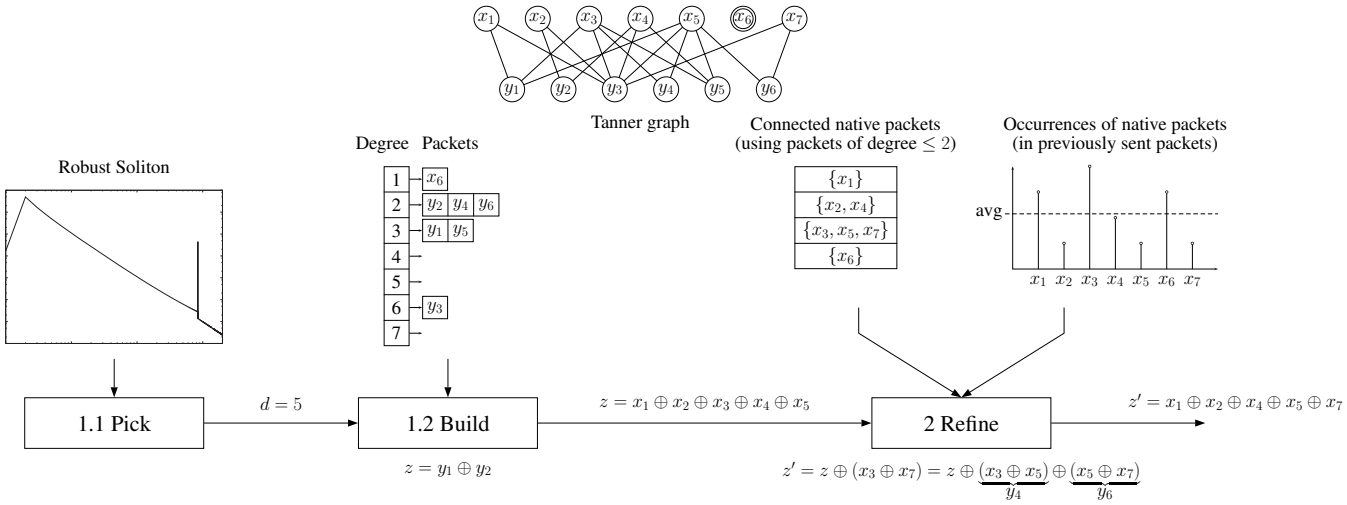


Figure 4. Overview of LTNC ($\mathbf{k = 7}$).

its degree is therefore $d(z) = 5$, that is the degree picked in the previous step. Steps 1.1 and 1.2 ensures that the degrees of fresh encoded packets sent by a node follow a Robust Soliton distribution as specified by LT codes.

Finally, p refines the encoded packet z built from the previous steps in order to decrease the variance of the distribution of degrees of natives packets in previously sent encoded packets. This step substitutes some native packets in the fresh encoded packet built in the previous steps (i.e., z here) with other ones that appeared less frequently in previously encoded packets, without jeopardizing the degree of z. This results in a fresh encoded packet z'. In LTNC, this is achieved with the help of encoded packets of degree 1 and 2. Effectively, if a native packet x appears in an encoded packet z and a native packet x' does not appear in z, then, adding the packet of degree 2 $x \oplus x'$ to z boils down to substituting x' to x in z (since $x \oplus x = 0$ and $z \oplus 0 = z$). Note that such an operation does not change the degree of the encoded packet. In the example of Figure 4, the native packet x_3 appears in more previously sent encoded packets than x_7 (this information is available from a dedicated data structure that gathers statistical information on previously sent encoded packets: the number of occurrences of each native packet) and appears in z while x_7 does not appear in z. x_3 is therefore replaced with x_7 by adding $x_3 \oplus x_7$ to z. This steps relies on the ability of the node to build $x_3 \oplus x_7$. To this end, each node maintains a specific data structure to determine which encoded packets of degree two it can build from the available ones. More specifically, a node maintains a partition of native packets where two native packets x and x' are in the same set if $x \oplus x'$ can be generated using only encoded packets of degree 2. In the example of Figure 4, x_3 and x_7 are in the same set since $x_3 \oplus x_5$ and $x_5 \oplus x_7$ are available (y_4 and y_6 in the Tanner graph).

It can be seen from this concrete example that the recoding method of LTNC involves several difficult algorithmic problems. For instance, how to determine if, for a given degree d, the node is able to build an encoded packet of degree d and how to select encoded packets to combine to reach that degree. The next section presents algorithmic solutions to the aforementioned problems and describes in detail the data structures and the associated maintenance techniques used by LTNC.

Table I
COMPLEMENTARY DATA STRUCTURES USED BY LTNC.

Data structure	**Purpose**
Encoded packets by degrees	find a set of encoded packets to build a fresh one of a given degree
Connected native packets	determine packets of degree 2 that can be built using only degree 1 and 2 encoded packets
Occurrences of native packets	determine substitutions of native packets that decrease the variance of degrees

B. Recoding LT encoded packets

In the following, we detail the different steps involved in the recoding method of LTNC.

1) Picking a degree: To match a Robust Soliton distribution of degrees for encoded packets sent, a node building a fresh encoded packet first picks a *target degree* d at random drawn from this specific distribution. However, the target degree may not be *reachable*, i.e., no packet of degree d can be built from the set of encoded and decoded packets available at the node. Assuming that a fresh encoded packet of degree d is built only from decoded native packets and encoded packets of degree lower than d (i.e., the building method does not leverage collisions, which is the case of LTNC as explained in the next paragraph), LTNC uses two heuristics to detect if a degree d is unreachable.

First, a degree d is unreachable if $\sum_{i=1}^{d} i \cdot n(i) < d$, where $n(i)$ is the number of encoded packets of degree i available at the node. For instance, the maximum reachable degree of a fresh encoded packet built from the set of encoded packet $\{x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_3, x_2 \oplus x_5\}$ is $2 \times 2 + 3 = 7$.

Second, the maximum reachable degree is upper-bounded by the number of native packets that either are decoded or appear in at least one encoded packet of degree less than d. For instance a packet of degree 5 cannot be generated from the set of encoded packets $\{x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_3, x_2 \oplus x_5\}$ since any linear combination of these encoded packets involve only 4 different native packets (i.e., x_1, x_2, x_3 and x_4).

If a picked degree is classified as unreachable (i.e., larger than one of the two upper bounds), a new one is picked and so on and so forth until the picked degree is accepted (i.e., not classified as unreachable). Note that this allows to discard immediately *some* unreachable degrees, however this does not guarantee that the picked degree is effectively reachable. For instance, none of the two aforementioned bounds would discard degree 3 for the set $\{x_1 \oplus x_2, x_3 \oplus x_4\}$ while this degree cannot be effectively reached. Similarly, a picked degree of 4 is not discarded for the set $\{x_1 \oplus x_2, x_2 \oplus x_3, x_4\}$. In our simulations, the first picked degree is accepted in 99.9% of the cases and the average number of retries (when the first degree is discarded) is 1.02.

2) Coping with a picked degree: This steps takes as input a picked degree d, a set $\mathcal{X}$ of decoded native packets and a set $\mathcal{Y}$ of encoded packets and builds a fresh encoded packet of degree d. Formally, the problem writes: given d, $\mathcal{X} = \{x_i\}_{i \in I}$ and $\mathcal{Y} = \{y_i\}_{i=1}^{k'}$, find $\mathcal{W} \subset \mathcal{X} \cup \mathcal{Y}$ so that $d(z) = d$, where $z = \bigoplus_{w \in \mathcal{W}} w$. The problem of finding a set of packets so that the sum of the degrees is exactly d is known as the *subset sum* problem which is NP-complete. The fact that the degree of the sum of two encoded packets may not be the sum of their respective degrees (e.g., the degree of $(x_1 \oplus x_2) \oplus (x_2 \oplus x_3)$ is 2 and not 4, this is called a *collision*) makes this problem even more difficult.

LTNC finds a sub-optimal solution in a greedy fashion, preventing collisions that decrease the degree of the fresh encoded packet being built. It examines the encoded packets ordered by decreasing degrees starting from d. A packet is added to the fresh encoded packet being built if the degree of the resulting encoded packet (i.e. the sum) *(i)* is increased and *(ii)* remains lower or equal to d . The algorithm uses a specific data structure, namely an index $\mathcal{S}$ of packets grouped by degrees (i.e., $\mathcal{S}[1] = \mathcal{X}$ and $\mathcal{S}[i]$ is the set of encoded packets of degree i in $\mathcal{Y}$ for $i > 1$). The degree of the resulting fresh encoded packet is lower than d. A pseudo-code version is given in Algorithm 1. In our simulations, the building step reaches the target degree 95% of the time and the average relative deviation to the target degree (i.e., target degree - obtained degree / target degree) is 0.2%.

In the example depicted in Figure 4, z has been obtained using Algorithm 1: the building algorithm starts with encoded packets of degree 3 since there is no packet of degree 4 or 5. It picks y_1 at random and adds it to z. y_5 is then examined and discarded as it would decrease the degree of z (i.e., $y_1 \oplus y_5$ is of degree 2). The algorithm then moves to encoded packet of degree 2 and picks y_2 at random. The encoded packet $z \oplus y_2 = y_1 \oplus y_2$ is of degree 5, y_2 is thus added to z and no more packets are further added. Effectively, as soon as $d(z) = d$, the condition $d(z) < d(z \oplus y) \leq d$ cannot be satisfied anymore.

Algorithm 1 Building an encoded packet of a given degree

Input: d ▷ Target degree
Output: z ▷ Fresh encoded packet with a maximum degree of d

```
1:  z ← 0                                          ▷ Fresh encoded packet being built
2:  i ← d
3:  S' ← S[i]
4:  while d(z) < d and i > 0 do
5:      if S' = ∅ then                  ▷ If there is no more packets of degree i
6:          i ← i − 1                                          ▷ Move to S[i − 1]
7:          S' ← S[i]
8:      else
9:          y ← pickAtRandom(S')
10:         S' ← S'\{y}
11:         if d(z) < d(z ⊕ y) ≤ d then
12:             z ← z ⊕ y
13:         end if
14:     end if
15: end while
```

3) Refining an encoded packet: This step *refines* the fresh encoded packet z obtained from the previous step by replacing some native packets with less frequent ones in order to decrease the variance of the degree distribution of native packets. This information is available from a specific data structure that stores, for each native packet, the number of occurrences in the previously sent encoded packets. The data structure is updated every time a fresh encoded packet is sent.

As explained above, a native packet x can be replaced with x' (denoted $x \sim x'$) if $x \oplus x'$ can be generated. In LTNC, refinement is achieved using only decoded native packets and encoded packets of degree 2: it is considered that $x \oplus x'$ can be generated if *(i)* x and x' are decoded or *(ii)* $x \oplus x'$ is available or *(iii)* there exists a third native packet x'' such that $x \sim x''$ and $x'' \sim x'$. By construction, the relation $\sim$ is an equivalence. Its equivalence classes correspond to the connected components in the graph where the vertices are the k native packets and there exists an edge between x and x' if $x \oplus x'$ is in the Tanner graph).

This information is available from a dedicated data structure cc that maps a native packet to an integer so that $x \sim x' \Leftrightarrow \mathrm{cc}(x) = \mathrm{cc}(x')$. The value $\mathrm{cc}(x)$ can be thought of as the index of the *leader* of the connected component. Initially, $\mathrm{cc}(x_i)$ is set to i for all $1 \leq i \leq k$. The data structure is then dynamically updated as follows: when a native packet x is decoded $\mathrm{cc}(x)$ is set to 0, when an encoded

packet of degree 2, say $x \oplus x'$, is received (or obtained by belief propagation during the process of decoding) $\mathrm{cc}(x'')$ is set to $\mathrm{cc}(x)$ for all x'' so that $\mathrm{cc}(x'') = \mathrm{cc}(x')$. This enables LTNC to determine in $\mathcal{O}(1)$ if an encoded packet of degree 2 can be generated. Figure 5 returns to the example depicted in Figure 4 and gives a leader-based representation of the connected components of native packets (using encoded packets of degree 1 and 2). When the encoded packet $x_3 \oplus x_4$ is received, both $\mathrm{cc}(x_4)$ and $\mathrm{cc}(x_2)$ are updated to $\mathrm{cc}(3) = 5$.

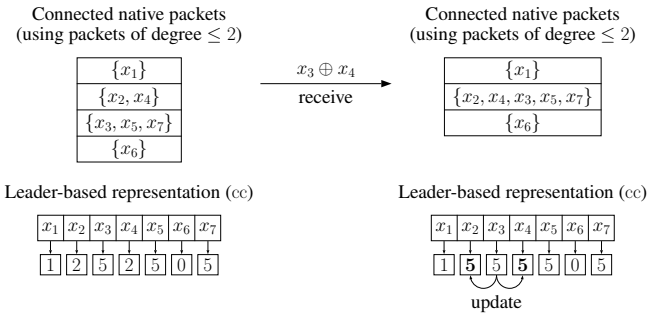


Figure 5. Leader-based representation of the connected components of native packets.

Using these two data structures, LTNC replaces iteratively each native packet x in z with the less frequent native packet x' that verifies the following three properties: (1) $x \sim x'$; (2) x' is less frequent than x; (3) z does not contain x'. This results in a refined fresh encoded packet z' that gives the minimum (for a given fresh encoded packet z to refine and a set of encoded packets of degree 1 or 2) variance of occurrences of native packets. Algorithm 2 gives a pseudo-code version of the refinement step.

Algorithm 2 Refining an encoded packet

Input: z ▷ Fresh encoded packet
Output: z' ▷ Refined fresh encoded packet

1: $z' \leftarrow z$ ▷ Fresh encoded packet being built
2: **for each** $x \in z$ **do**
3: $\mathcal{A} \leftarrow \{x''$ s.t. $x \sim x'$ and $x'' \notin z'$ and x'' is less frequent than $x\}$
4: **if** $\mathcal{A} \neq \varnothing$ **then**
5: $x' \leftarrow \arg\min_{x'' \in \mathcal{A}} \mathrm{frequency}(x'')$
6: $z' \leftarrow z' \oplus (x \oplus x')$
7: **end if**
8: **end for**

In the example depicted in Figure 4, z' has been obtained using Algorithm 2: x_1 cannot be replaced with any other native packet (i.e., it is the only native packet in its connected component); x_2 is less frequent than any other native packet; x_3 can be replaced with x_7 (since $x_3 \sim x_5$ and $x_5 \sim x_7$), x_7 is less frequent than x_3 and x_7 is the least frequent such native packet: x_3 is therefore replaced with x_7; x_4 and x_5 are not replaced since they are less frequent than x_3 (i.e., which is the only native packet not contained in z' at this stage).

The efficiency of the refinement algorithm highly relies on the fact that, due to the Robust Soliton distribution used in LT codes, more than half of the encoded packets are of degree 1 or 2, thus resulting in a high refining power. In our simulation, the relative standard deviation (standard deviation /average) of the number of occurrences of native packets in encoded packets sent is 0.1%.

C. Optimizations

With random linear codes, including LT codes, encoded packets have a low – but non-zero – probability of being non-innovative (i.e., packets that can be generated from other encoded packets already available at the node). In Random Linear Network Coding (RLNC), a partial Gaussian reduction step detecting non-innovative packets is performed when a fresh encoded packet received is inserted in the data structures (i.e., the code and data matrices). However, in LTNC, belief propagation does not provide immediate detection of non-innovative packets. This results in some redundancy in the data structures.

As both memory and CPU usage are related to the number of encoded packets stored at a node, redundant packets should be avoided. To this end LTNC includes a low-complexity redundancy mechanism to detect and remove non-innovative encoded packets upon reception or during the process of decoding. The detection mechanism can be adapted for systems where a feedback channel is available allowing the sender and the receiver to communicate and agree on the encoded packet to send. This results in an increased convergence speed and a decreased bandwidth usage since: *(i)* the encoded packets sent are more likely to be innovative for the receiver, and *(ii)* if the sender detects that it cannot generate an innovative packet for the receiver, no packet is sent.

1) Detecting redundancy: A packet is said redundant or non-innovative for a node if it can be generated from the encoded packets available at the node. Considering an encoded packet $z = \bigoplus_{i \in I} x_i$, z can be built without collision if there exist two sets J and J' so that: $J \cup J' = I$ and both $y = \bigoplus_{j \in J} x_j$ and $y' = \bigoplus_{j \in J'} x_j$ can be generated (or are available at the node). Using this recursive formalization under the non-collision assumption, a large proportion of non-innovative packets can be detected. However, the complexity increases exponentially with the degree of the encoded packet being checked, even when using dynamic programming. Interestingly enough, when using LT codes, most encoded packets circulating in the system have a low degree. Applying this redundancy detection mechanism to low degree encoded packets allows discarding a large proportion of non-innovative encoded packets at low cost. Moreover, high-degree packets are less likely to be non-innovative. Therefore, detecting non-innovative packets of high-degree is useless in most of the cases.

To reduce complexity of the redundancy detection mechanism, in LTNC it is applied only to encoded packets of degree less than or equal to 3 (that is almost two thirds of the encoded packets with Robust Soliton). Furthermore, we improve on the algorithm described in the previous paragraph by taking into account collisions for encoded packets of degree 2. Detection makes use of the connected components of native packets. An encoded packet of degree 1 is redundant if it is available at the node. An encoded packet $y = x \oplus x'$ of degree 2 is redundant if x and x' are in the same connected component (i.e., $\mathrm{cc}(x) = \mathrm{cc}(x')$). Algorithm 3 gives a pseudo-code version of the redundancy detection mechanism used in LTNC. Note that the redundancy detection mechanism can be applied on encoded packets stored which degree drops to 3 during the process of decoding. For instance, in the example depicted in Figure 4 the node stores an encoded packet $y_5 = x_3 \oplus x_4 \oplus x_5$. If the node somehow decodes x_4, x_4 will be propagated to y_5 which incurs a xor operation. This operation is useless, as it would give $x_3 \oplus x_5$ which can be generated from the other encoded packets. The redundancy mechanism of LTNC prevents such useless operations.

Algorithm 3 Detecting redundant packets: $\mathrm{isRedundant}()$

Input: y ▷ Encoded packet of degree ≤ 3
Output: b ▷ Returns true if y can be generated and false otherwise

if $d(y) = 1$ **then** ▷ y is a native packet $y = x$
  $b \leftarrow \mathrm{isDecoded}(x)$
else if $d(y) = 2$ **then** ▷ $y = x \oplus x'$
  $b \leftarrow (\mathrm{cc}(x) = \mathrm{cc}(x'))$
else if $d(y) = 3$ **then** ▷ $y = x \oplus x' \oplus x''$
  $b \leftarrow (\mathrm{isRedundant}(x) \wedge \mathrm{isRedundant}(x' \oplus x'')) \vee$
    $(\mathrm{isRedundant}(x') \wedge \mathrm{isRedundant}(x \oplus x'')) \vee$
    $(\mathrm{isRedundant}(x'') \wedge \mathrm{isRedundant}(x \oplus x')) \vee$
    $\mathrm{isAvailable}(x \oplus x' \oplus x'')$
end if

Determining if a packet of degree 1 or 2 is redundant can be done in $\mathcal{O}(1)$ operations. Assuming the use of a complementary structure allowing $\mathcal{O}(\log k)$ lookups of encoded packets of degree 3 (e.g., a binary search tree), the redundancy detection mechanism of LTNC is $\mathcal{O}(\log k)$. In our simulations, this mechanism decreases by 31% the number of redundant encoded packets inserted in the data structure upon reception.

2) Preventing redundancy: We now assume the existence of a feedback channel allowing the receiver to provide the sender with useful information helping it to increase the probability of sending an innovative encoded packet.

Consider in a first step a basic binary feedback channel that allows the receiver to abort the transfer if the encoded packet sent is detected as a redundant one. Assume for instance that an encoded packet is sent through a TCP connection and that the corresponding code vector precedes the data. Then, as soon as the code vector is received, the receiver can run the redundancy detection mechanism and close the connection if the packet is non-innovative. This prevents the sender from wasting bandwidth sending useless data.

Consider now, a fully operational feedback channel allowing the receiver to provide the sender with more complex information. A naive solution is to run the very same redundancy detection algorithm at the sender (assuming that the required information available at the receiver has been transfered using the feedback channel beforehand) and abort the transmission if the fresh generated packet is not innovative for the receiver. In this case the bandwidth is saved but the *session* is wasted as no packet is sent. A more sophisticated solution is to determine the intersection of what can be generated at the sender and what is innovative for the receiver. Again, this is a difficult problem in general, but simple and efficient solutions can be implemented for low degree encoded packets at a reasonable computational cost. In LTNC, a "smart" packet construction algorithm using information from the receiver is used only for packets of degree 1 and 2 in order to limit the amount of data exchanged: the leader-based representation cc_r is sent to the sender through the feedback channel. Note that similar information can be partially obtained or inferred in a wireless setting by snooping packets sent by close nodes as in COPE [6]. For degree 1 or 2, the problem can be formalized as follows:

$(d = 1)$: Find x s.t. $\mathrm{isAvailable}_s(x)$ and $\mathrm{not}(\mathrm{isAvailable}_r(x))$
$(d = 2)$: Find x, x' s.t. $\mathrm{cc}_s(x) = \mathrm{cc}_s(x')$ and $\mathrm{cc}_r(x) \neq \mathrm{cc}_r(x')$,

where cc_s (resp. cc_r) is the leader-based representation of the connected components of native packets at the sender (resp. the receiver).

The first case is straightforward. The second can be addressed by constructing iteratively a mapping σ between the connected components of native packets at the source and at the receiver. The native packets are processed in random order. If a connected component at the source overlaps with two components at the receiver (i.e., maps to two distinct components), an innovative packet can be generated. Algorithm 4 gives a pseudo-code version of the smart packet construction algorithm.

Algorithm 4 Constructing an innovative packet $(d = 2)$

$\sigma\{0, \ldots, k\} \mapsto (\perp, \perp)$ ▷ Mapping between connected components

for each $x_i \in \{x_i\}_{i=1}^{k}$ **do**
  $(j, x) \leftarrow \sigma[\mathrm{cc}_s(i)]$
  if $j = \perp$ **then** ▷ First time component $\mathrm{cc}_s(i)$ is visited
    $\sigma[\mathrm{cc}_s(i)] \leftarrow (\mathrm{cc}_r(i), x_i)$
  else if $j \neq \mathrm{cc}_s(i)$ **then** ▷ Already visited with different mapping
    $\mathrm{send}(x \oplus x_i)$
  end if
end for

Consider the example depicted in Figure 6 and assume

that native packets are processed by increasing indexes. When x_1 is processed, $\sigma[\mathrm{cc}_s(x_1)]$ (i.e., $\sigma[1]$) has not been initialized yet and is thus set to $(\mathrm{cc}_r(x_1), x_1)$ (i.e., $(7, x_1)$). Similar updates are performed for x_2 and x_3. When processing x_4, a mapping already exists and is consistent with $\mathrm{cc}_r(x_4)$ (i.e., the packet $x_2 \oplus x_4$ is not innovative). However, with x_5 the mapping is not consistent: an innovative packet (i.e., $x_3 \oplus x_5$) can therefore be generated.

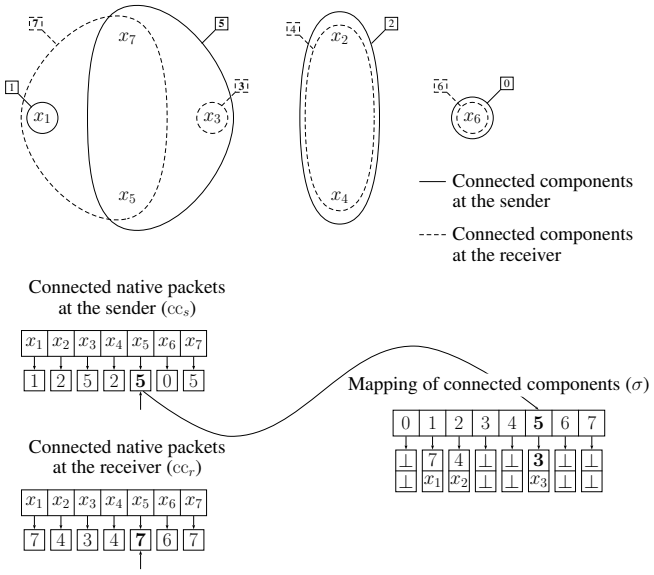


Figure 6. Sample execution of the "smart" packet construction algorithm: component 5 at the sender overlaps with components 3 and 7 at the receiver.

IV. Evaluation

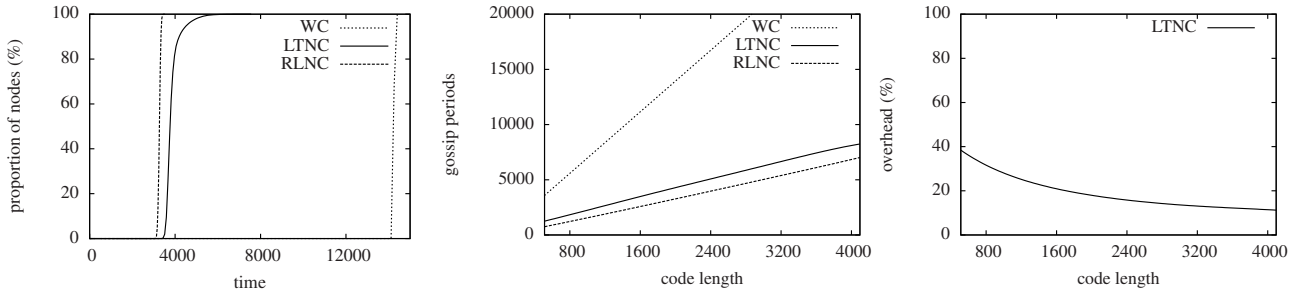
In this section we present an evaluation of LTNC based on simulations. In order to evaluate LTNC, we compared it against two dissemination schemes: without coding and with random linear network coding (RLNC). Our simulations show that LTNC incurs only 20% more message emissions than RLNC while reducing the computational complexity by up to 99% at decoding. In addition, despite the fact that LTNC does not perform as well as RLNC with respect to latency, it still outperforms epidemic schemes that do not use network codes, thus preserving the benefits of using coding techniques.

A. Experimental setup

We consider a network of N nodes where a content is disseminated from a source to all of the N nodes in an epidemic fashion. The content is divided into k native packets of size m. The code vectors of encoded packets, represented by bitmaps, are included in the headers of the packets. The source periodically injects encoded packets in the network. As soon as a node has received more than a given proportion of the packets, it starts periodically generating and pushing fresh encoded packets. The proportion of packets required to trigger recoding is controlled by a parameter of the system called *aggressiveness*. In our simulations, the aggressiveness is set so that the completion time is minimized (typically 1% for LTNC, note that in WC and RLNC, recoding can be done without delay). Packets are pushed to nodes picked uniformly at random in the network, using an underlying peer sampling service (e.g., [23]). The set of nodes to which a node pushes packets is renewed periodically in a gossip fashion. The underlying overlay is therefore *dynamic*. Communications are *unicast* and we assume the existence of a *binary feedback channel* allowing the receiver to abort the transfer of a native or encoded packet detected as non-innovative. Aborting a transfer is simply achieved by closing the TCP connection (assuming that code vectors are in the headers of the packets, the receiver is able to determine if the corresponding packet is redundant before the content is actually sent). In such an application, the size of the system N is generally *a few thousands* of nodes, and a typical content is a file of 512 MB (e.g., a video) divided into $k = 2,048$ blocks of size $m = 256$ KB.

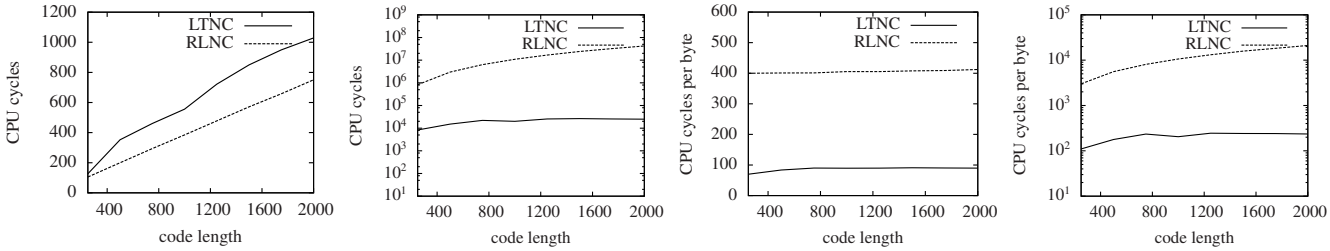
We compare LTNC against two reference schemes, namely Random Linear Network Coding (RLNC) and Without Coding (WC), that capture the trade-off between the performance with respect to content dissemination (i.e., average time to complete and communication cost) and the computational cost of the operations performed at the nodes (i.e., recoding and decoding). In our simulations, we therefore implemented our proposed approach (LTNC) and the two reference schemes:

- **LT Network coding (LTNC):** In this scheme, the degree distributions of encoded and recoded packets and the distribution of native packets follow the distributions of LT codes. Linear combinations of native packets are performed over GF(2). The recoding and redundancy detection techniques used are those described in Sections III-B and III-C. Decoding is performed using the belief propagation algorithm.
- **Without Coding (WC):** In this scheme, no coding is used. Therefore nodes exchange only native packets and detecting a non-innovative packets boils down to checking if the packet has already been received. Nodes buffer the innovative packets they receive up to a fixed number b. If the buffer is full, the oldest packet is discarded. Each received innovative packet is forwarded to f nodes (unless the packet is removed from the buffer). At each gossip period *one* buffered packet (typically the one that has been sent the least number of times) is sent to *one* random node. It has been shown that f must be greater than $\lceil \ln N \rceil$ to ensure with high probability that all nodes eventually receive all the native packets [24].
- **Random Linear Network Coding (RLNC):** In this scheme, nodes generate fresh encoded packets by lin-



(a) Convergence (b) Average time to complete (c) Overhead

Figure 7. Dissemination performance.



(a) Recoding (control) $\times 1000$ (b) Decoding (control) (c) Recoding (data) (d) Decoding (data)

Figure 8. Computational cost of each operation (CPU cycles).

early combining, over GF(2), random combinations of previously received encoded packets. The number of encoded packets involved in the recoding operation is bounded by a given parameter, namely the *sparsity* of the codes, set to $\ln k + 20$. Limiting the number of encoded packets combined to generate a fresh encoded packet decreases the computational cost of the recoding operation without impacting the performance of content dissemination. This set of parameters is widely acknowledged as the optimal setting for linear network coding [7], [8]. Detecting non-innovative packets and decoding are performed using Gauss reduction.

B. Experimental results

We now compare the performance of the three dissemination schemes presented in the previous paragraph. Experimental results have been averaged over 25 Monte-Carlo simulations. We evaluate LTNC, RLNC and WC along three metrics: *(i)* convergence time, *(ii)* overhead, and *(iii)* computational complexities.

Convergence: Figure 7a plots the proportion of nodes that were able to decode the k native packets as a function of time. The network is composed of $N = 1,000$ nodes and the file disseminated is composed of $k = 2,048$ packets of size $m = 256$ KB. Results show that the convergence using LTNC is slightly slower than using RLNC. Yet, LTNC largely outperforms WC, showing the benefit of coding schemes in content dissemination. This is confirmed by the results depicted in Figure 7b where the average time to complete for several values of the code length k ranging from 512 to $4,096$ is plotted. Interestingly enough, the time overhead of LTNC with respect to RLNC decreases with the code length k.

Overhead: Figure 7c depicts the communication overhead for several values of the code length k ranging from 512 to $4,096$. For $k = 2,048$, LTNC sends 20% more packets than necessary. The communication overhead decreases with k. Note that without coding and with RLNC the communication overhead is null as the redundancy detection mechanism can abort *all* transfers of non-innovative packets using respectively lookups and Gauss reductions.

Computational cost: Figure 8 compares the computational complexities in terms of CPU cycles of LTNC and RLNC. The complexity of the operations performed on the control structures (e.g., Tanner graph for LTNC and code matrix for RNLC) and those on the data are plotted separately for both recoding and decoding. This results have been obtained on an Intel Xeon 32bit at 2.33GHz with 1GB of RAM. The program has been compiled with gcc 4.4 with the optimization parameter set to `-O3`.

We observe that for $m = 256$ KB the cost of the operations performed on the control structure is negligible as compared to those performed on data. Due to the building and refining steps used by LTNC, its complexity at recoding is higher than for RLNC. However, since the average degree of encoded packet sent is lower for LTNC, the cost of recoding data is lower for LTNC. Note that in both cases, the recoding complexity on data scales well

with the code length. For RLNC this is due to the use of sparse codes as described in the experimental setup. For $k = 2,048$, LTNC decreases the decoding complexity by more than 99%, thanks to belief propagation (made possible by the structure of our distributed LT network codes), which fully justifies the reasonable communication overhead for time constrained applications. In conclusion, LTNC advantageously trades communication overhead for computational complexity. More specifically, the increase (20%) of the number of packets sent is largely compensated by a huge gain in CPU cycles at decoding (99%).

V. Related Work

To the best of our knowledge, LTNC is the first coding scheme to tackle the problem of generating encoded packets that match a given degree distribution using *encoded packets* which makes it the first LT network coding scheme.

Linear network coding is an efficient paradigm introduced in [1] that allows reaching max flow using path diversity in a communication network. However to achieve optimal performance it requires to determine the recoding matrices applied at each intermediary nodes function of the global network topology which is difficult in a decentralized setting. Random linear network codes [21] (RLNC) alleviate the need for global coordination between intermediary nodes by generating at each node random linear combinations of previously received encoded packets. RLNC achieves close-to-optimal performance and has been successfully applied to content dissemination in sensor networks [25] and file sharing systems, namely Avalanche [3]. However it has been widely criticized for the high computational cost of the decoding [7], [8]. Some optimizations to reduce the complexity of Gauss reduction – such as generations [13] – have been proposed but the complexity remains quadratic with the length of the codes.

LT Codes [10] and Raptor Codes [26] (LT codes built on precoded native packets) are erasure codes that rely on specific degree distributions of encoded packets to allow low-complexity decoding using belief propagation. In their initial versions, they do not provide a recoding method to use them as network codes. In [9] a network coding solution based on Raptor codes is proposed: sources nodes send Raptor-encoded packet and intermediary nodes generate fresh encoded packets using specific recoding matrix. However, this recoding technique does not preserve the degree distribution of Raptor codes. Therefore, the decoder must perform a high complexity Gauss reduction thus loosing the benefit of belief propagation.

Distributed LT Codes [27] distributes the construction of LT encoded packets on several *server nodes*. The output of the server nodes are then summed up by a front-end machine, namely the *relay node* and injected in the network. Intermediary nodes then just forward encoded packets received: no network coding is involved. The main contribution of the paper is to derive an appropriate distribution of degrees for the packets generated at each server node such that the degree distribution of packets emitted by the relay node fits a Robust Soliton distribution.

In [28], stacked LT codes for dissemination trees are proposed: encoded packets at level i in the tree are LT codes built over encoded packets at level $i-1$ (i.e., considering level $i-1$ encoded packet as native packets). The problem of collecting, at a sink node, data initially stored at networked sensor nodes is addressed in [11] using Growth codes: the degree of the encoded packets circulating in the network grows with time (initially native packets are sent) such that each encoded packet received enables to decode a new native packet. This solution addresses a different problem than LTNC and requires the nodes to be loosely synchronized. Furthermore, the use of Growth codes was motivated by the fact that the authors want to maximize the number of native packets decoded in a many-to-one scenario while LTNC aims at maximizing the proportion of nodes that can recover *all* the native packets in a one-to-many scenario. It is shown in [11] that Growth codes outperform LT codes in the first case but perform worse than LT codes in the latter.

Network codes have been widely used in distributed storage as well to ensure data persistence. LTCDS [5] (LT Codes Distributed Storage) replicates on all the nodes – in an encoded form – k native packets, each of them being initially available at a single node. The native packets circulate in the network using random walks and each node uses them to build the encoded packet it stores: LT codes are built using only native packets. In the end, the degree distribution of the encoded packets distributed in the network follows a Robust Soliton distribution. Similar solutions have been proposed in [12] and [4].

In [29], the author derives the probability of an encoded packet to be innovative, function of the number of native packets decoded at the receiver. Deriving this distribution allows the sender to optimize the utility of the encoded packet sent by carefully choosing its degree. The information on the number of native packets decoded by a node is available from an Oracle. An extension of this work alleviating the need for an Oracle is proposed in [30]. It provides algorithmic tools to evaluate the similarity between the set of encoded packets available at the sender and the receiver. Similarity is estimated using a Bloom filter that summarize the content available at both nodes with a small fixed number of bits. Information about similarity is then used to choose an optimal degree (i.e., that maximizes the probability of being innovative) for the encoded packet sent. However, the two aforementioned techniques address neither the problem of generating an encoded packet of a given degree from a set of encoded packets nor the problem of matching a given degree distribution of native packets.

VI. Conclusion

In this paper, we presented novel low complexity network codes (LTNC) based on LT codes, initially proposed in erasure coding approaches. LTNC provides an attractive alternative to random linear codes for they greatly simplify the decoding complexity, trading the Gaussian reduction for belief propagation. LTNC implements a set of algorithms enabling to recode from encoded packets. This is achieved by preserving on the fly and in a fully-decentralized manner the statistical properties of LT codes required to benefit from belief propagation decoding.

Experimental evaluations show that LTNC consistently outperforms an unencoded dissemination protocol with respect to delay, preserving the benefit of coding. Random Linear code are optimal and not surprisingly LTNC introduces a small overhead in term of latency and communication. However, LTNC substantially reduces, up to 99%, the decoding complexity when compared to random linear codes. This significantly broadens the spectrum of application settings for network codes, typically where nodes have low capabilities (e.g., sensors). The application framework of wireless sensor networks is especially attractive as the broadcast nature of the communication medium opens many perspectives of further optimizations. In addition, we believe that, beyond epidemic content dissemination applications, LTNC can be used in self-healing distributed storage systems.

References

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network Information Flow," *IEEE Transactions On Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[2] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *INFOCOM*, 2005.

[3] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding," in *IPTPS*, 2006.

[4] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks Through Decentralized Erasure Codes," in *IPSN*, 2005.

[5] S. A. Aly, Z. Kong, and E. Soljanin, "Fountain Codes Based Distributed Storage Algorithms for Large-Scale Wireless Sensor Networks," in *IPSN*, 2008.

[6] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the Air: Practical Wireless Network Coding," in *SIGCOMM*, 2006.

[7] M. Wang and B. Li, "How Practical Is Network Coding?" in *IWQoS*, 2006.

[8] G. Ma, Y. Xu, M. Lin, and Y. Xuan, "A Content Distribution System Based on Sparse Network Coding," in *NetCod*, 2007.

[9] N. Thomos and P. Frossard, "Raptor Network Video Coding," in *MV*, 2007.

[10] M. Luby, "LT Codes," in *FOCS*, 2002.

[11] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth Codes: Maximizing Sensor Network Data Persistence," in *SIGCOMM*, 2006.

[12] S. Aly, Z. Kong, and E. Soljanin, "Raptor Codes Based Distributed Storage Algorithms for Wireless Sensor Networks," in *ISIT*, 2008.

[13] P. Maymounkov, N. J. A. Harvey, and D. S. Lun, "Methods for Efficient Network Coding," in *Allerton*, 2006.

[14] 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.

[15] D. Charles, K. Jian, and K. Lauter, "Signature for Network Coding," Microsoft Research, Tech. Rep. MSR-TR-2005-159, 2005.

[16] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Médard, "Resilient Network Coding in the Presence of Byzantine Adversaries," in *INFOCOM*, 2007.

[17] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An Efficient Signature-based Scheme for Securing Network Coding against Pollution Attacks," in *INFOCOM*, 2008.

[18] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. O. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," in *INFOCOM*, 2007.

[19] A. Duminuco and E. Biersack, "A Pratical Study of Regenerating Codes for Peer-to-Peer Backup Systems," in *ICDCS*, 2009.

[20] M. Mitzenmacher, "Digital Fountains: A Survey and Look Forward," in *ITW*, 2004.

[21] 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*, vol. 52, no. 10, pp. 4413–4430, October 2006.

[22] R. M. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[23] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, p. 8, 2007.

[24] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, "Epidemic Information Dissemination in Distributed Systems," *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[25] J. Widmer and J.-Y. Le Boudec, "Network Coding for Efficient Communication in Extreme Networks," in *WDTN*, 2005.

[26] A. Shokrollahi, "Raptor Codes," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 2551–2567, Jun. 2006.

[27] S. Puducheri, J. Kliewer, and T. E. Fuja, "Distributed LT Codes," in *ISIT*, 2006.

[28] R. Gummadi and R. Sreenivas, "Relaying a Fountain Code Accross Multiple Nodes," in *ITW*, 2008.

[29] J. Considine, "Generating Good Degree Distributions for Sparse Parity Check Codes using Oracles," CS Department, Boston University, Tech. Rep. BUCS-TR-2001-019, 2001.

[30] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 767–780, 2004.