Data dissemination in distributed systems using rateless codes

Univerisitá degli Studi di Torino

Dipartimento di Informatica

Dottorato di Ricerca in Informatica
XXIV ciclo

PhilosophiæDoctor (Ph.D.) thesis

Valerio Bioglio

Advisor:

Prof. Marco Grangetto

Ph.D. Coordinator:

Prof. Mariangiola Dezani

Abstract

Erasure codes are a possible solution for the information reliability problem in random networks. Novel encoding techniques, such as Random Network Coding and Rateless Codes, are emerging in this field, but the introduction of new ideas brings new challanges that must be addressed. For example, encoding the information to be spread introduces delays caused by the time required to decode the data upon receipt. These codes are also sub-optimal because slightly more data is requested in order to retrieve the initial information. Moreover, the more they are near to the optimality bound, the more their decoding complexity increases.

This thesis proposes a series of new encoding and decoding algorithms able to diminish the complexity of these codes, while approaching the optimality bound. After a theoretical analysis of the proposed techniques, they are analyzed in various applications for content distribution in peer-to-peer networks, distributed storage systems and network management and monitoring.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

LIST OF TABLES

Introduction

Recent results have pointed out that rateless codes can be profitably exploited to improve the performance (by avoiding the need for content reconciliation at the receiver) and reliability (by providing robustness with respect to peers churning) of data distribution in networks organized in random overlays, as peer-to-peer (P2P) networks or Wireless Sensors Networks (WSN). The use of coding techniques increases the reliability of content distribution applications such as distribution of bulk data, application level multicast, P2P streaming applications (1, 2, 3, 4) and efficient broadcasting in ad hoc wireless networks (5, 6), to name a few. Most cited results have been catalyzed by the seminal promises of network coding (7, 8), where nodes in the network are allowed to combine information. The deployment of network coding at the application level, e.g., in the field of P2P file sharing or video streaming, has been limited. This is primarily due to of the added computational cost associated with linear coding. Nowadays, such complexity issues must be carefully reconsidered, as a novel class of erasure codes, known as rateless codes (9, 10), designed for application level coding, is becoming a practical tool for efficient coded data dissemination.

Rateless codes (also called Fountain codes) are a family of erasure codes where the rate, i.e. the number of coded and transmitted symbols, can be adjusted on the fly. These differ from standard channel codes that are characterized by a rate, which is selected in the design phase. A rateless encoder can generate an arbitrary number of coded symbols. This indicates that rateless codes perform well where the erasure probability is not known as in multicast applications, where the encoder outputs into a shared medium which cannot tune its transmissions to an individual receiver. The approach used to transmit such codes is called Digital Fountain (DF), since the transmitter can be viewed as a fountain emitting coded symbols until all the interested receivers (the sinks) have received the number of symbols required for successful decoding.

The encoding process of fountain codes is simple: given k input symbols (for simplicity, we assume that the input and output symbols are bits) and a discrete probability distribution P over [1,k] (called degree distribution), the encoder generates each output symbol choosing a random degree d according to P, and then choosing uniformly at random a set of d input symbols and XORing them. The classical decoding algorithm for these codes is Message Passing (MP): given n output symbols, the decoder selects one of degree one. The value of the corresponding input is set, and that input is then cancelled out of all other outputs it is a part of. Decoding stops when there are no degree-ones left. This algorithm is very fast, but usually needs a large number of output symbols to enable decoding. The decoding process can also be seen as the resolution of a system of equations, so Gaussian Elimination (GE) can be used. GE requires less input symbols to execute the decoding. Moreover GE performance is far less sensitive to the parameters chosen for the degree distribution, making the code design simpler. Alternatively, if the cost of an algorithm is computed as the number of operations needed to decode then GE complexity is larger then MP.

The degree distribution is a crucial component of the design of a Fountain code. Luby Transform (LT) codes (9) are the first class of efficient rateless erasure codes that achieve optimum performance as the data length increases. In (9) the Robust Soliton Distribution (RSD) $\tau(c, \delta)$ is proposed. In the RSD, c is a suitable positive constant and δ is the allowed failure probability at the decoder. It is demonstrated that the decoder fails to recover the data with a probability of at most δ from a set of $K = k+O(\sqrt{k} \times ln^2(k/\delta))$ coded symbols, which means that successful decoding is attained with $K = k(1 + \epsilon)$ with $lim_{k\to\infty}\epsilon = 0$, i.e., the code is asymptotically optimal.

Other known Fountain codes are Raptor codes (13), Windowed codes (11) and Growth codes (12).

Rateless codes can solve the complexity problem raised by network coding, enabling the creation of practical communication schemes that use linear encoding techniques. However, many problems still remain. The overhead of rateless codes is usually only asymptotically optimal: for small values of k, mainly used in real-time applications, the overhead could be very high (larger than 10%). Conversely, some rateless codes can maintain a small overhead even for small values of k, at the cost of an increase in the decoding complexity. The tradeoff between overhead and complexity is a core point in the design phase of an encoding scheme.

An important property of Network Coding is the ability to construct a distributed encoding of the information. The information is not only encoded at the source, but also all the nodes of the network support the encoding of the packets, thus obtaining a real distributed encoding scheme. In rateless codes this is not completely possible. Many authors faced this problem, proposing different solutions for various network configurations, without really solving the problem ((50), (38), (49)): the encoding of rateless codes is inherently centralized. The possibility of encoding the information using rateless codes in a distributed way is still a problem.
Another problem, common for network coding and rateless codes, is the delay of the transmission due to the decoding of the packets. In classical distribution systems, when the information is received, it is ready for use: conversely, encoded schemes add a further delay in order to retrieve the plain information. Decoding algorithms that are able to decode the packets early must be studied in order to address this issue. Incremental Decoding and Partial Decoding are two paradigms created in order to decrease the decoding delay.

Incremental Decoding. The MP algorithm (as it is usual for all decoding algorithms) has to wait to receive the first $k + \epsilon$ output symbols to attempt the decoding process. Moving forward, the decoder attempts decoding at a prescribed set of times. In general, at every decoding attempt decoding algorithms start "from scratch" without using the soft information produced in the previous decoding attempt. Alternatively, Incremental Decoding (ID) algorithms continue from the decoding results of the last decoding attempt (14). This means that the ID computational effort is distributed over all symbol receptions, because they keep decoding while waiting for the next symbols. Classical decoding algorithms spend almost all their computations after the required number of symbols has been received, and the actual decoding time is the sum of the time spent to receive the symbols plus the time spent to decode them.

Partial Decoding. The design of rateless codes has been optimized so that the low-complexity decoder can recover all the inputs provided it starts with slightly more outputs than inputs. Partial decoding concerns the intermediate performance of the codes, i.e. the case when the number of received output symbols is less than the number of input symbols. In this case it is not possible to fully recover all

input symbols, but it is important to know the fraction of input symbols that can be recovered as a function of the number of received output symbols and the probability distribution used to generate the code.
In application, the intermediate performance of a code can be very important. For example, consider a scenario where a data stream is to be transmitted to multiple users over a shared channel. The stream needs to be decoded in real time. If rateless codes are used for the transmission, the stream would have to be broken into blocks of symbols, and each block would be encoded (and decoded) separately. In this case, it is possible that some user may not receive the requisite number of output symbols for some input block. It is reasonable, however, that many real-time applications can be reliably played back from a large enough fraction of the inputs. Thus it is of interest to optimize the intermediate performance of a code. Sanghavi (17) has theoretically studied the intermediate performance of rateless codes using the MP decoder.

This work is divided into two parts. In the first part, a theoretical solution for some of the problems of network coding and rateless codes is presented. Specifically, we propose an algorithm for decoding Fountain Codes using Gaussian Elimination called On-the-fly Gaussian Elimination (OFG) (15). OFG is an incremental decoding algorithm that builds a triangular matrix by exploiting every received symbol starting from the very first one. This enables the OFG to halve the number of operations needed to decode w.r.t. GE. We also propose the Optimal Partial Decoding (OPD) algorithm (34). This algorithm is able to decode the maximum number of input symbols when given an arbitrary set of output symbols. To date, no results exist for such an algorithm; to the best of our knowledge, this is the first optimal partial decoding algorithm for rateless codes.
Later we study the distributed encoding of rateless codes, and in particular the recoding of pre-encoded packets by the nodes, as proposed in the Random Network Coding (RNC) schemes. However, a blind combination policy leads to an increase in decoding complexity: the degree of the packets tends to degenerate from the inital degree distribution of the used rateless code to Dense Codes, a family of higher-complexity rateless codes. To cope with this problem, we propose Band Codes, a new family of rateless codes that are able to simultaneously maintain the decoding complexity and

keep overhead under control.
Finally, we propose a general framework based on the Digital Fountain approach to create regenerating codes (22) that is able to reconstruct the lost fragments by contacting a number of storage nodes lower than the number of fragment used to retrieve the message. Moreover, the proposed framework is able to increase the reliability of the information creating new encoded fragments without retrieving the plain message.

In the second part we use the aforementioned theoretical tools to attack various problems that involve the dissemination of data in a distributed network. Particularly, we study the application of rateless codes in three subfields: *content distribution*, *distributed storage systems* and *network management and monitoring*.

In content distribution, the objective is to explore how coding theory can provide benefits to existing applications in terms of performance, resilience and efficiency. We also emphasize hostile environments with frequent node mobility and communication errors such as isolated mobile internet worlds and wireless mesh networks. Existing state-of-the-art employing coding for content distribution is abundant, particularly in peer-to-peer networks. The nature of such networks makes them particularly suitable for coding, as their ability to operate in a distributed fashion and scale, and self-organize in the presence of a highly transient population of nodes, computer and network failures. Data is usually divided in packets. In this field, one of the main problems is content reconciliation due to the high probability to receive duplicated packets.
Content distribution solutions provide a mechanism for distributing content on the Internet in order to maximize bandwidth usage and to improve accessibility and reliability (18, 19). A typical content distribution solution relies on placing dedicated equipment at certain places inside or at the edge of the Internet. These Content Distributed Networks (CDNs) improve network performance by maximizing the resource utilization through content replication, caching, request routing, server-load balancing, etc (20). While they have the advantage of good maintainability of the system, they are often vulnerable to single points of failure. Techniques such as content replication, caching and mirroring are widely used by centralized CDNs to alleviate these problems (19).
Peer-to-peer networks facilitate the formation of autonomous networks by being more flexible to the dynamic changes in the network. The network can thus spontaneously adapt to the demand by taking advantage of the resources provided by every node. The system capacity theoretically grows at the same rate as the demand, creating

limitless scalability for a fixed cost (21). P2P distributed computer architectures are designed for the sharing of computer resources (content, storage, CPU cycles) by direct exchange, rather than requiring the support of a centralized server or authority. P2P architectures are characterized by their ability to adapt to failures and to accommodate transient populations of nodes while maintaining acceptable connectivity and performance. In a traditional P2P content distribution scheme, the server splits the file of interest into blocks; next, peers download the blocks from the server and also collaborate to distribute downloaded blocks among themselves. Each node can recover the original file after downloading all the components. Such approaches are sensitive to sudden departures of nodes, and their performance is affected by the policy used to determine which block to forward. Nonetheless, using codes in a P2P system reduces these problems and increases the robustness to losses.

We develope a protocol that is able to significantly improve the performance of coded data dissemination obtaining low average delays and efficient utilization of upload and download peer bandwidths. Low average distribution times and high bandwidth utilization are simply obtained by letting peers saturate their upload bandwidth, forwarding useful coded packets as soon as possible. Of course,a forwarding activity that is too aggressive may result in an unacceptable high overhead due to the amount of duplicated packets received from neighbors. To reduce the amount of duplicated packets, we first evaluate the effect of throttling the speed used by peers to saturate the available upload bandwidth. We then show that combining useful packets with packets accumulated during the decoding process results in a lower probability of forwarding duplicated packets, reducing the overhead.

However, this policy leads to an increase of the decoding complexity. In large networks, the system tends to degenerate from the inital degree distribution of the used rateless code to Dense Codes. We propose a recoding algorithm to the nodes that, when applied to Band Codes, permits a strict control on the complexity of the decoding to cope with this problem.

In the field of distributed storage systems, our objective is to study the impact of Fountain Codes in distributed storage, video and other content for their ability to provide maximum throughput and minimum latency. Existing works show that traditional distributed storage techniques, such as erasure codes, achieve good reliability, by optimizing the trade-off between reliability and redundancy. In practical distributed

storage systems, however, (22) shows that other considerations enter into play, such as network bandwidth consumption. In such settings, (23) shows that traditional schemes alone do not suffice.

Distributed storage systems are becoming the de-facto method of data storage for the new generation of applications. As an example, modern cloud applications developed by companies like Amazon, Google and Yahoo! that require terabytes of data, need to rely on distributed computing and storage to meet availability, scalability and performance demands. Compared to traditional relational database systems, distributed storage networks increase storage efficiency and data availability by providing shared storage access to computers and servers in multiple locations (24). Distributed storage systems provide reliable access to data through redundancy spread over individually unreliable nodes. The geographic distribution of resources also results in a lower observed latency, and resources can be placed closer to clients. A distributed storage system consists of a set of storage elements placed into nodes, which function independently of each other and thus exhibit independent failure patterns. These nodes are often connected through a network with arbitrary topology, and the information objects are stored in specific nodes according to a mapping function. Typically, one wants to reliably recover the information in a distributed storage system with the lowest communication cost possible, and also by requiring the lowest possible aggregated storage.

A distributed storage system can be easily conceived as the process of dissemination and recovery of information taking place across several nodes, and thus it is a good candidate to employ some form of coding. By encoding information, distributed storage systems increase reliability over former non-coded methods. We develope an encoding protocol for a distributed storage system using rateless codes.

Finally, we explore network management and monitoring, a crucial tool to monitor the health of a network where network coding has been extensively used with great success. The ability to control and to infer network characteristics accurately and efficiently, without incurring in significant use of costly network infrastructure, is critical. This is particularly true in volatile wireless environments where resources and infrastructure support are very limited.

A critical network engineering functionality is to monitor the health of a network and

adapt its use to the demands of the end users. In order to perform this network monitoring, control mechanisms are needed to infer network characteristics efficiently and accurately, without significant use of costly network measurement infrastructure. This process is referred to as network tomography. We develope a multicast protocol that is able to spread the information owned by a peer to all the other peers in the network. Using a C++ simulator, we studied how rateless codes can improve the distribution of information about the health of a peer in the network.

Part I

Theory

Chapter 1

Rateless Codes

1.1 Erasure Channel

Many channel models exist in the information theory field. The model we want to focus on is the Packet Erasure Channel (PEC), an extension of the Binary Erasure Channel (BEC) where the binary alphabet is extended to strings of bits, i.e. a BEC where the transmission is performed on a per packet basis instead of per bit.

The communication scheme of an erasure channel is showed in Fig.1.1. A symbol (generally expressed with 0 and 1) has to be transmitted from a sender X to a receiver Y. The channel loses the symbol with probability p_e, called the *erasure probability*, or transmits the symbol correctly with probability $1 - p_e$. In other words, the transmitted symbol can get lost through the channel, or can arrive at the receiver unchanged. This communication channel model can be used to model transmissions in an IP network; the best effort strategy of these networks, where the correctness of the content of a packet is assured by the lower-level protocols but the packets are discarded due to routing or congestion problems, is the practical application of a PEC channel.

A solution for the transmission problem over a PEC is the use of a feedback channel from the receiver to the sender, wich is used to inform the sender about lost packets. Alternatively, a system of acknowledgements can be used: every time a packet is received, the receiver sends a message to the sender. However, these messages travel over the same PEC channel, and hence can get lost; a protocol like TCP can manage this problem, provinidg a good solution to the transmission problem over a PEC. The main problem of TCP is its high transmission overhead, i.e. the large number of acknowledge

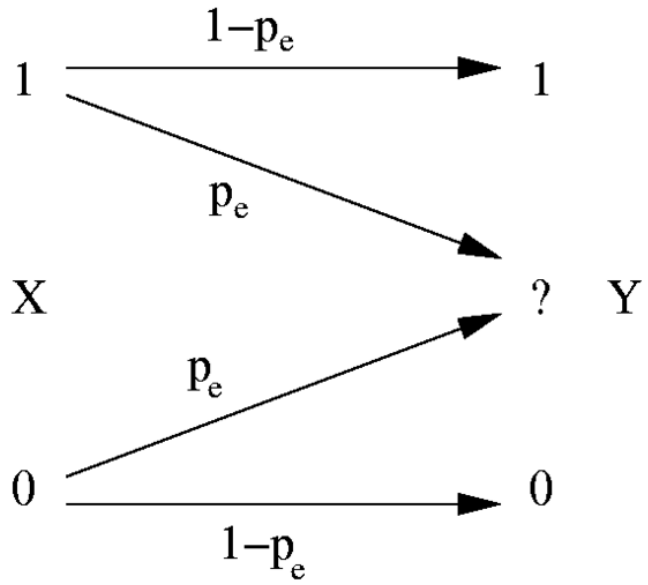


Figure 1.1: Binary Erasue Channel (BEC) model

messages that need to be sent at every packet transmission.
Another solution to the transmission problem is to use an *erasure code*, that is a forward correcting code specifically designed for BECs. An erasure code transforms a message m of k symbols (bits or packets) into an encoded message of n symbols. The ratio $\frac{k}{n}$ is fixed and depends on the code used. It is called the *rate* of the erasure code. The n symbols are sent over the BEC and the receiver receives $n' \leq n$ of them. In an erasure code, it is called *overhead* of the code the number $k' \geq k$ of symbols required to recovery m; k' can depend on the subset of the received symbols. An optimal erasure code has the property that $k = k'$; in coding theory, such a code is called a Maximum Distance Separable (MDS) code. The classic MDS codes are Reed-Solomon (RS) codes. The main disadvantage of RS codes is their high computational complexity, which only makes them practical for small values of k and n.

Another problem with RS codes, which is common to all the classic erasure codes, is the fixed rate. The rate of the code must be chosen by the sender according to the erasure probability. This assures that the required number of symbols will be received with a certain probability. However, it is possible that the erasure probability of the

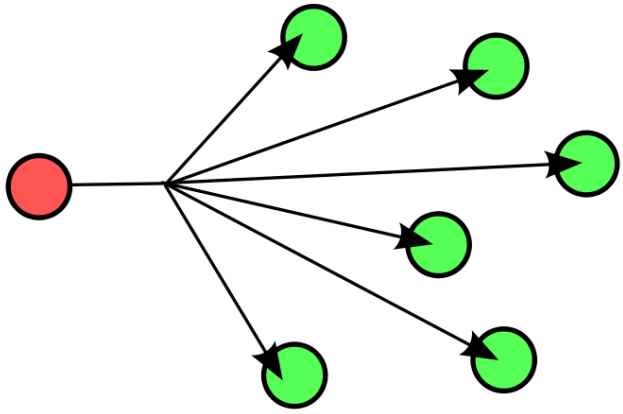


Figure 1.2: Broadcast transmission model

channel is unknown or may change during the transmission. Hence, the number of symbols received by the receiver is lower than k. In this case, the receiver needs more symbols in order to recover m. These new symbols cannot be created on-the fly, and a new erasure code must be used. Another problem associated with the fixed rate erasure codes is the broadcast transmission (Fig. 1.2). If a message has to be transmitted by broadcasting to a certain number of receivers, and each receiver is connected to the sender through an erasure channel with different erasure probability, the sender should use a different erasure code for each channel, not using the broadcast property. As an alternative, the sender could encode the message using an erasure code for the worst channel and send the same encoded symbols to all the receivers. However, this is not an optimal solution because high rate codes would be used for low erasure probability channels.

To face these problems, Luby et al. (33) propose a new approach: the Digital Fountain.

1.2 Fountain Codes

The idea behind Fountain Codes is simple: the encoder is seen as a (digital) fountain that emits a continuous stream of drops, the encoded packets. When a receiver wants to receive the message, he metaphorically holds a bucket under the fountain, collecting a number of drops a little larger than k. The receiver can ask for any subset of size

a little larger than k in order to recover m. The other name of fountains codes is derived from this characteristic: these codes are also called *rateless codes* (RC). This name comes from their ratleless nature: in fact, the rate, which is dependent on the number of encoded packets created by the code, is theoretically equal to zero. This is because the stream of packets emitted by the fountain is endless. Using these codes, it is possible to create new encoded packets on the fly, solving the main problem of fixed rate erasure codes. However, these codes are suboptimal, as a certain overhead $k' = (1 + \epsilon)k$ is requested in order to recover the message.

Rateless codes are a good idea, and it is possible to build them. The most simple version we can create are Dense Codes (DC).

Consider a message m divided into k packets $m_1, \dots m_k$, where a packet is a data unit composed of l bits that can be transmitted over a PEC. The total size of m is $k \cdot l$ bits. At every transmission possibility, the sender creates an encoded packet c_n as follows: first, a random element e_n of $GF(2)^k$, called *equation*, is extracted. This is a random string of bits of length k where every bit is a 1 or a 0 with a probability of $\frac{1}{2}$. The encoded packet is generated as the exclusive-or (xor) of the packets of m that lay in a position i for which e_n has a 1 in that position:

$$c_n = \sum_{j=1}^{k} e_{n,j} \cdot m_j$$

where $e_{i,j}$ is the j-th element of equation e_i and the summation is performed in $GF(2)$, i.e. modulo 2. The generating matrix E of the code can be created from the equations, as the matrix formed by the equations as rows. Therefore, the encoding matrix is a random $k \times n$ binary matrix. The encoded packets are transmitted to the receiver along withtheir equations. The packets can get lost during the transmission: let us suppose that the receiver receives n' packets among the packets sent by the sender. If G is the $k \times n'$ matrix formed by the received equations as rows, G is a sub-matrix of the generating matrix E. In order to recover the plain message, if c is a vector formed by the received packets, the receiver has to solve the system of equations

$$G \cdot m = c.$$

This system is solvable if G contains k linearly independent rows. This means that if $n' < k$ the system is unsolvable and the receiver needs more packets in order to recover

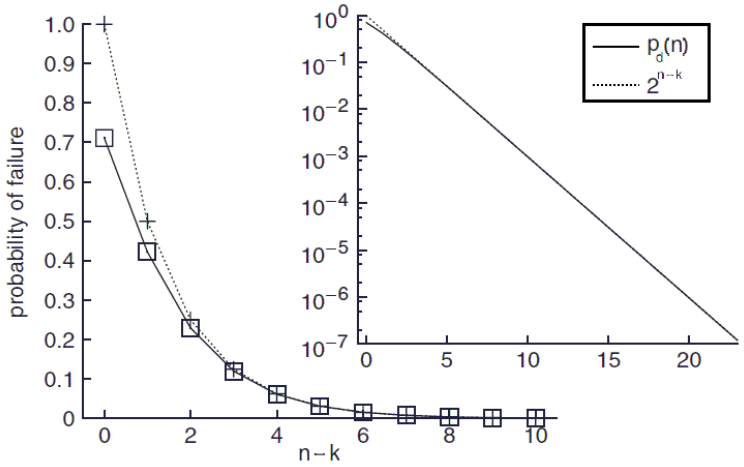


Figure 1.3: Performance of Dense Codes (from (10))

m. This is not a problem for a rateless code: the sender can create new encoded packets on the fly and send them to the receiver until the plain message is completely recovered. On the contrary, if $n' \geq k$, the system can be solved by inverting G (in particular, the sub-matrix of G formed by k linearly independent rows). Now a new problem arises, concerning the probability $p_l(n)$ that a random $k \times n'$ binary matrix will contain k linearly independent rows. In the case of a random $k \times k$ matrix, this problem has a simple solution; $p_l(k)$ is the product of $k-1$ probabilities. The first factor is given by the probability that the first row is not the all-zero row, that is $(1-2^{-k})$. The second factor is the probability that the second row is different from both the first row and the all-zero row, given by $(1-2^{-(k-1)})$. In general, factor i is the probability that row i is linearly independent from the first $i-1$ rows and different from the all-zero row; this means that row i has to be different from set of 2^{i-1} equations given by all the possible linear combinations of $i-1$ equations plus the all-zero row. Such a probability is $(1-2^{-(k-i+1)})$. Finally,

$$p_l(k) = \prod_{i=1}^{k}(1-2^{-i}).$$

This probability is approximately 0.289 for $k = 100$. If $n' > k$, the calculus is more complex. In figure 1.3 the failure probability $p_d(n)$ is plotted for $k = 100$. This probability can be over-bounded by the function 2^{k-n}, represented in the figure by the dotted line. From the figure it is possible to note that even if Dense Codes are not

optimal, they are near-optimal: the failure probability decreases exponentially with the number of extra packets received. This means that the expected overhead of the code, i.e. the expected number of extra packets required for the recovery, is about $\sum_{i=1}^{\infty} \frac{i}{2^i} = 2$, which is a very low number.
However, the main problem of Dense Codes is the large computational complexity of their decoding; the expected decoding cost depends on the inversion of a random binary matrix, which takes about $O(k^3)$ operations. For large values of k, this complexity turns out to be too computationally expensive. Similarly, their encoding complexity is $O(k^2)$. A solution with lower computational cost would be preferred: this solution is provided by the LT codes (9).

1.3 LT Codes

LT codes are the first family of low-complexity rateless codes. The main idea of the LT codes is to maintain the generating matrix as sparse as possible; indeed, many studies on the invertibility of binary random matrices show that it is possible to maintain high probability of inversion even with sparser matrices. Moreover, in (9) the author proposes a low complexity decoder for his code, thus solving another problem posed by Dense Codes.
The encoder is similar to the encoder of DC: the main difference is the generation of the equations. For LT codes, the degree d_n of the equation e_n is chosen first; this degree is chosen randomly from an appropriate degree distribution $\rho(d)$. After that, the equation is chosen at random among the degree-d_n equations in $GF(2)^k$. The degree distribution of LT codes is the crucial point of the process, and it will be discussed later. As an example, a DC can be seen as a rateless code with a binomial distribution $B(k, \frac{1}{2})$ as the degree distribution. The remainder of the encoding process is the same for the DC: the encoded packet c_n is calculated as

$$c_n = \sum_{j=1}^{k} e_{n,j} \cdot m_j.$$

The decoder is especially designed for decoding low-density codes. In fact, the decoding process can be reduced in the resolution of a system of equations $G \cdot m = c$, where, for LT codes, G is a sparse matrix. In this case, Luby proposes to solve the system

using the Message Passing (MP) algorithm. This algorithm is very simple and fast: it is performed when at least k packets have been received by the receiver. At every cycle, a row G_u that contains a unique 1 in column j (i.e. an already decoded packet) is found. The column j is then checked: every time a 1 is found at row i, that row and the corresponding element of c are xored with G_u and c_u respectively, obtaining $G_i = G_i \oplus G_u$ and $c_i = c_i \oplus c_u$. This elimination is performed for all the rows that contain a 1 in column j: at the end, column j will have a unique 1 in correspondence of row u. That row and its correspondent element of c are considered decoded, obtaining $m_j = c_u$, and G_u and c_u are then substituted by all-zero vectors. After that, the cycle terminates and a new cycle can begin. This algorithm ends in two cases: when G turns out to be all zero, hence the decoding process is successful, or when no degree-one rows are found, and in this case the decoding attempt fails.

The possibility of always finding new degree-one rows during the process is of paramount importance for the MP algorithm. The degree distribution of LT codes is designed to keep the expected number of degree one rows equal to 1 at every iteration. In (9) the author proves that, theoretically, the best distribution is the Ideal Soliton Distribution, which is defined as:

$$\rho(d) = \begin{cases} \frac{1}{k} & if \quad d = 1 \\ \frac{1}{d(d-1)} & if \quad 1 < d \leq k \\ 0 & otherwise \end{cases}$$

Unfortunately, this distribution performs poorly in practice because of the large variance for the probability of finding degree-one rows during the MP decoding process. To solve this problem, Luby proposes the Robust Soliton Distribution (RSD), originated by Ideal Soliton Distribution with two parameters added.

The Robust Soliton Distribution uses the parameters c and δ in order to ensure that the expected number of degree-one rows during the MP decoding process is about $S = c \log_e(k/\delta)\sqrt{k}$. Using these parameters, the auxiliary function

$$\tau(d) = \begin{cases} \frac{S}{kd} & if \quad 1 \leq d \leq \lfloor \frac{k}{S} \rfloor - 1 \\ \frac{S}{k} \log_e(\frac{S}{\delta}) & if \quad d = \lfloor \frac{k}{S} \rfloor \\ 0 & otherwise \end{cases}$$

is calculated. Finally, the RSD $\mu(d)$ is calculated as

$$\mu(d) = \frac{\rho(d) + \tau(d)}{\sum_{i=1}^{k} \rho(i) + \tau i}.$$

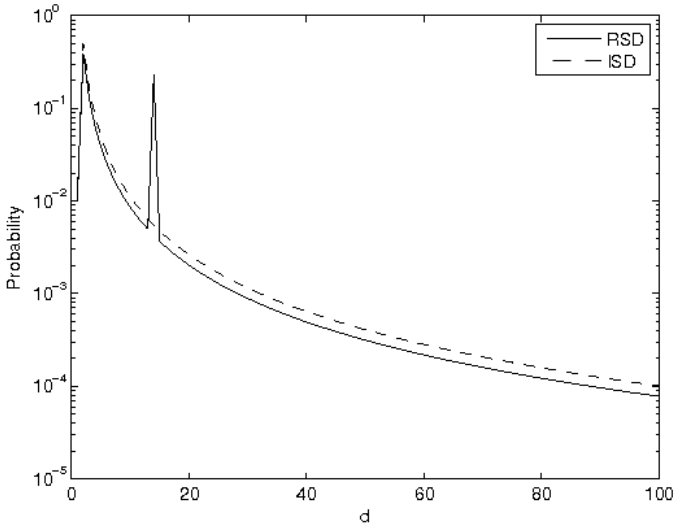


Figure 1.4: Comparison between ISD and RSD(0.1,0.1) for $k = 100$

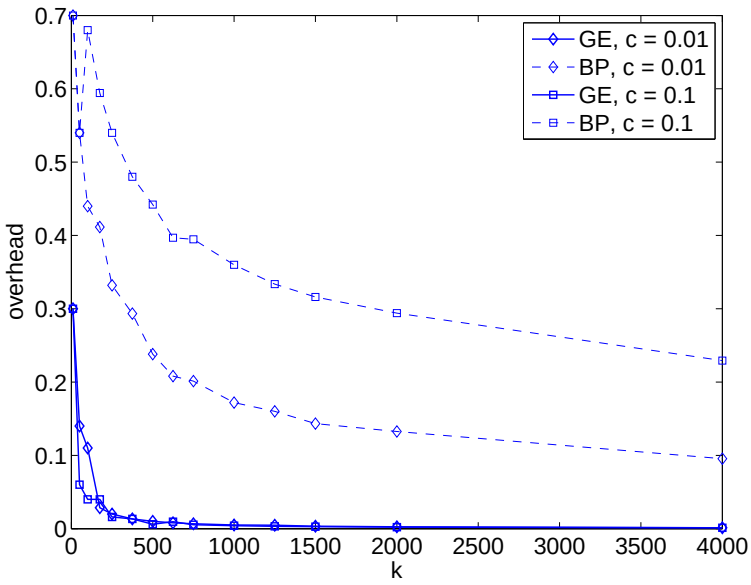


Figure 1.5: Average overhead using $RSD(c, 0.01)$ distribution

In fact, the RSD is similar to the Ideal Soliton distribution with the exception of a peak for $d = \lfloor k/S \rfloor$, that ensures a larger connection between rows (see Fig. 1.4).
Finally, Luby proves that the overhead of LT codes is vanishing for large values of k; in particular, he proves that $k' = k + 2\log_e(S/\delta)S$ received packets can recover the message with a probability of at least $1 - \delta$; if $k' = k(1 + \epsilon)$, and ϵ is the overhead of the code, $\epsilon \to 0$ for $k \to \infty$. However, the convergence speed also depends on the employed decoding algorithm. As seen earlier, the LT decoding process can be viewed as the resolution of a system of equations, and the classic algorithm for the resolution of this problem is the Gaussian Elimination (GE) algorithm. In Fig. 1.5, the overhead ϵ is shown versus k for GE and BP decoders for a LT code with $\delta = 0.01$ and $c = 0.01$. It can be noted that, for GE, the overhead converges faster than for BP; in general, GE-like algorithms requires fewer encoded packets to decode. Moreover, GE performance is far less sensitive to the parameters chosen for the degree distribution, making the code design simpler. On the other hand, if we compute the cost of an algorithm as the number of row xor and swap operations needed to decode, then BP complexity, that is $O(k \log k)$, outperforms GE complexity, that is approximatively $O(k^2)$.

1.4 More Rateless Codes...

The coding and decoding computational complexity of LT codes scale as $k \log k$; however, many other rateless codes were created after the Luby's seminal paper.
The most known rateless codes apart from the LT codes are the Raptor codes (13). These codes have surprising linear coding and decoding computational complexity. This property is obtained through a pre-encoding step. The initial message, divided into k packets, is initially encoded, using an excellent fix-rate outer erasure code, in $k' = k/(1 - f)$ pre-encoded packets. These k' pre-encoded packets are then encoded a second time with a weak LT code. This ensures, with high probability, that it is possible to recover $(1 - f)k' = k$ packets when slightly more than k packets are received; these packets are now decoded using the outer code decoder, recovering the initial message.
Lesser known rateless codes are Windowed codes (WC) (11), that will be very useful in this thesis. The main idea of these codes is to create equations in which the ones are concentrated in a window of length w. In particular, the authors recommend the creation of equations of degree $\sigma = \lceil 2log_e k \rceil_{odd}$ and window length $w = 2(\sqrt{k}-1)(\sigma-1)/(\sigma-2)$.

In fact, the sender chooses a random integer $m \leq k$, that is the initial position of the window. After that, the equation is created by putting a 1 in position m and σ ones among the w positions that follow m; if the bottom of the equation is reached, the window will wrap back to the top. The decoding algorithm of these codes is Gaussian Elimination, instead of MP, due to the absence of degree-one equations.

Chapter 2

Incremental Decoding of Rateless Codes

2.1 Introduction

The "on the fly" nature of the encoding process of rateless codes is not reproduced in the decoding phase: typically, the receiver attempts to recover the plain message m only after the reception of k packets. If the decoding fails, the next attempt will be performed from scratch after a certain number τ of new packets are received. In fact, all the partially decoded packets obtained during decoding attempts are deleted and the process is repeated from the beginning every time. This is a waste of time and computation resources. It should be of interest to maintain the partially decoded information from one decoding attempt to the next. This is the basic idea of *incremental decoding*. In particular, we say that a decoding algorithm is incremental if the decoding process is spread during all the packets reception, i.e if, in case a decoding fails, the algorithm waits for new encoded packets, restarting the decoding process at the point where the previous attempt failed. In the case of LT codes, the most known incremental algorithms are derived directly from MP and GE. The one used more frequently is the incremental version of BP, called IBP (26). However, the Incremental Gaussian Elimination (IG) algorithm (25), which contains an incremental way to compute the triangularization step in GE without repeating it in case of decoding failure, could be used. In summary, these algorithms work as follows:

IBP algorithm: A row i_0 of G that contains only one 1 in column j_0 (a degree-one

row) is selected; it follows that $m_{j_0} = y_{i_0}$, all the 1s in column j_0 are canceled and their y_i are xored with y_{i_0}. The above process is iterated until the matrix G becomes an all-0 matrix (decoding success), or until no more degree-one rows can be found (decoding failure). In case of decoding failure, a new encoded packet is received, the 1s in the known positions of the corresponding equation are canceled and decoding is reattempted.

IG algorithm: The GE triangularization step is performed only once after k encoded packets (and their corresponding equations) are received. In case of failure, G is only partially triangular; IG tries to fill the "bad" rows (rows without a 1 on the diagonal) using new coded packets and the corresponding equations. This process is incremental in the sense that rows of G and the new equations are xored and swapped without repeating an expensive GE step on the whole matrix. As soon as G turns into triangular form, back substitution is performed to complete the decoding.

2.2 On the Fly Gaussian Elimination

On the Fly Gaussian Elimination (OFG) is a GE-like algorithm that does not wait for the first k packets to attempt the GE triangularization as in (25); rather it builds a triangular matrix G by exploiting every received packet starting from the very first one. Moreover, OFG employs a swap heuristic that yields a sparse triangular matrix reducing the cost of row xor and swap operations and the final back substitution. The main idea is to write matrix G in a triangular form as soon as possible by deleting redundant equations on the fly. Informally, OFG works as follows (all the details are presented in Algorithm 1): assume G is a partially triangular $k \times k$-matrix, i.e., either a row has leftmost 1 on diagonal or it is all-zero. Upon receiving an encoded packet y_i along with equation b_i we find the position of the leftmost 1 in b_i that we denote as s_i. If the row $G[s_i]$ is all-zero then we can replace $G[s_i]$ by b_i; otherwise, we xor b_i and $G[s_i]$, as well as y_i and y_{s_i}. We then obtain a new equation b'_i and a new coded packet y'_i. The new equation is such that the $s'_i > s_i$. The row finding and xoring are iterated until the equation is either placed into G or all 1s in the equation are cancelled, i.e., the equation is discarded. The number of iterations depends on the probability of a collision with a full row, that increases as the matrix is being populated. We experimentally observed that keeping a sparser G markedly decreases the number of iterations required to find

$$\begin{matrix}1&0&0&1&0\\0&0&0&0&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&0\end{matrix} \qquad \begin{matrix}1&0&0&1&0\\0&1&1&0&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&0\end{matrix}$$

(a) (b)

$$\begin{matrix}1&0&0&1&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{matrix} \qquad \begin{matrix}1&0&0&1&0\\0&1&1&0&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&1\end{matrix}$$

(c) (d)

Figure 2.1: triangularization step in the OFG algorithm.

the correct row in G for a given received packet, especially when G fills up. To this end we define the following swap heuristic that can be applied at any iteration: if the row $G[s]$ is full, i.e., it has its leftmost 1 on diagonal, but the equation to be inserted has a lower degree than $G[s]$; then the equation and $G[s]$ are swapped along with the corresponding coded packets.

After this triangularization phase, the source packets are computed by means of simple back-substitution. Ffor the sake of brevity this step is not included in Algorithm 1.

We provide a concept of how OFG works by considering a simple example. G is the partially triangular matrix in Fig. 2.1(a) and $b_1 = [01100]$ is the new received equation. In this case $s_1 = 2$, so we check if the $G[2]$ is full or not: it is not, so we can insert b_1 in that row, thus obtaining matrix G in Fig. 2.1(b). G is still not triangular ($G[5]$ is all zeros), therefore we need a new equation. Equation $b_2 = [11010]$ is received at this point. We get $s_2 = 1$, but $G[1]$ is already full; in this case b_2 and $G[1]$ are xored, obtaining the new equation $b_2' = [01000]$. We should insert this equation in $G[2]$, but it is full, so we should xor that row and b_2'. However, note that $G[2]$ has degree 2 while b_2' has degree 1: according to the proposed heuristic, we swap the row and the equation.

As a consequence we get $G[2] = [01000]$ and $b_2' = [01100]$. Now xoring $G[2]$ and b_2', we obtain $b_2'' = [00100]$ which should go in $G[3]$. This latter is already full, but its degree is greater than the degree of b_2''. Therefore, we repeat the swap and xor operations. The iteration of the algorithm operations leads to matrix G as depicted in Fig. 2.1(c). Note that without using the swap heuristic, we could triangulate G anyway, obtaining the matrix in Fig. 2.1(d). It is easily observed that the swap heuristic we devised yields a sparser triangular matrix. To summarize, the OFG algorithm triangulates G on the fly, by using row xoring and swap operation, while maintaining the sparsity of G.

Algorithm 1 On the Fly Gaussian Elimination

```
Initialize k × k-matrix G to 0
Initialize k-vectors Y and NumOnes to 0
Initialize EmptyRows = k
while EmptyRows > 0 do
  receive k-vector NewEq and encoded packet NewY
  s ← LeftmostOne (NewEq)
  EqOnes ← Degree(NewEq)
  while G[s][s] = 1 do
    if EqOnes ≥ NumOnes[s] then
      NewEq ← NewEq ⊕ G[s]
      NewY ← NewY ⊕ Y[s]
      s ← LeftmostOne(NewEq)
      PackOnes ← Degree(NewEq)
    else
      Swap(NewEq, G[s]), Swap(NewY, Y[s])
    end if
  end while
  if s < k then
    G[s] ← NewEq and Y[s] ← NewY
    NumOnes[s] ← EqOnes
    EmptyRows ← EmptyRows − 1
  else
    delete NewEq
  end if
end while
```

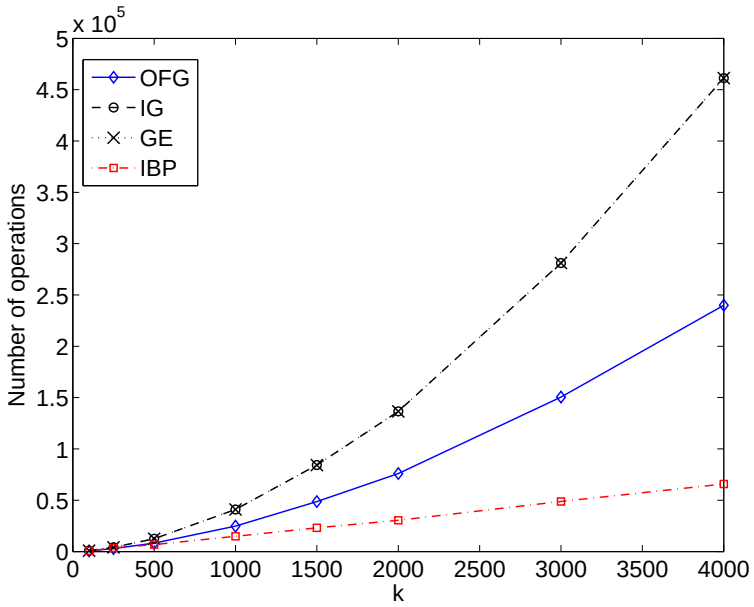


Figure 2.2: Complexity of triangularization step.

2.3 Simulation Results

We experimented with IBP, IG and OFG algorithms for several values of k using an LT code with $\delta = 0.01$ and $c = 0.01$. All the techniques have been implemented in C language using the same level of optimization in order to obtain fair comparisons. We ran 1000 encodings/decodings for each algorithm, and we presented averages of the relevant performance indexes.

In Fig. 2.2 we show the complexity of the triangularization step, which is computed by counting the total number of row xor and swap operations, for OFG, IG and IBP vs. k. For comparison, the cost of an ideal GE triangularization (performed only once as soon as G turns to full rank) is also reported. As shown in (25), the cost of IG is equal to that of a single GE triangularization, hence their curves in Fig. 2.2 are almost overlapped. Using OFG one halves the number of operations w.r.t. IG and GE while guaranteeing the same overhead. IBP is still faster than OFG because of its $O(k \log k)$ complexity. Furthermore, Fig. 2.3 shows the number of 1s in G after the triangularization phase vs. k for OFG, IG and GE. It can be noted that the proposed swap heuristic yields a sparser G in the OFG case. Moreover, OFG computational effort is distributed on all packet receptions. This feature turns out to be of paramount importance when taking

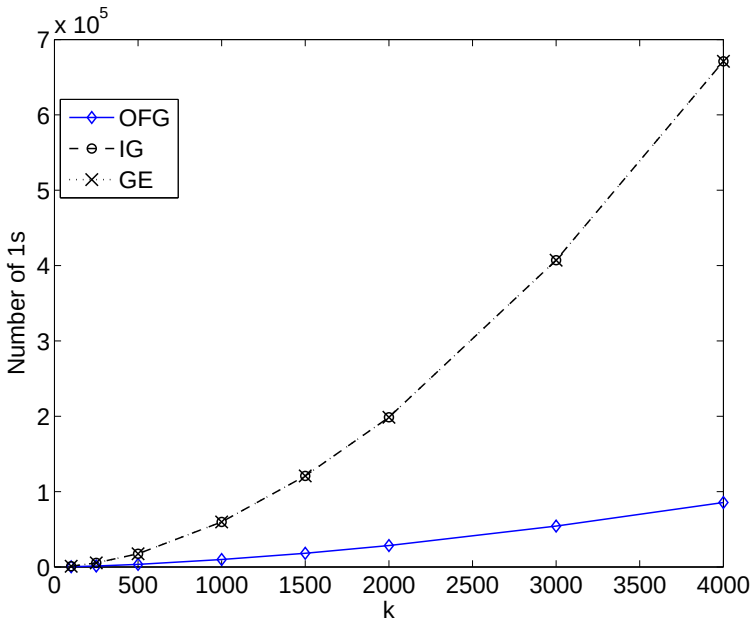


Figure 2.3: Number of 1s in the triangular matrix (excluding diagonal).

into account the packet reception delay. IBP and IG spend almost all the computations after the required number of packets has been received and the actual decoding time is the sum of the time spent to receive the packets plus the time spent to solve the system. On the contrary, OFG continues to triangulate the matrix while waiting for the next packets. This enable OFG to decode almost immediately after the last packet arrival, while IBP and IG are only beginning most of their decoding operations. Since the complexity to insert a row in G is $O(k)$, the total complexity of OFG is $O(k^2)$. In Fig. 2.4 we report the normalized (over k) number of operations per packet as a function of the percentage of received packets. It can be noted that the maximum value in Fig. 2.4 is about 0.2, equivalent to only $k/5$ operations per packet. Clearly, the row insertion cost increases with the number of received packets, since the number of empty rows is reduced. Finally, we measured the real decoding time with a multi-threading and concurrent implementation of the receiving and decoding tasks. The experiments have been worked out on a 2.66 GHz Intel Core Duo CPU equipped with 2 GB RAM. In Fig. 2.5 we show the CPU usage (%) as a function of the time, when the receiver downloads 1000 byte coded packets at 1Mbps. LT coding with $k = 10000$ is used. We observe that OFG allows for a 12.5% reduction in the overall decoding

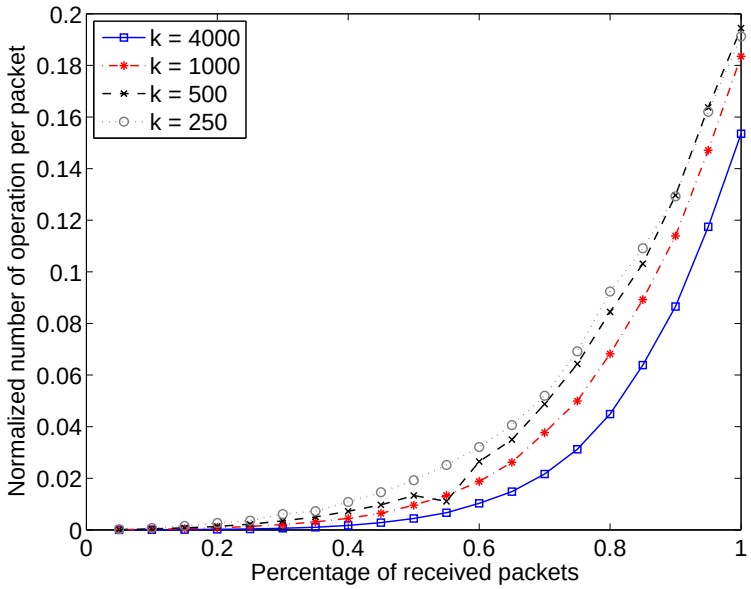


Figure 2.4: Normalized operations per packet vs. percentage of received packets.

time w.r.t. IBP and a more remarkable 28.9% w.r.t. to IG. IBP requires more packets to start decoding due to the larger overhead ($\epsilon = 0.06$). On the other hand, OFG is able to decode almost immediately after the arrival of the last packet, thus taking full advantage of the reduced overhead ($\epsilon = 5 \cdot 10^{-4}$) and exhibiting less intensive use of the CPU. IG has a limited overhead as well, but performs most of its operations at the end, making it the slowest solution in this scenario. In fact, the large peak for IG appears at $T = 80$ s, which corresponds with the time needed to download the first k coded packets, and it is mainly due to the first and unique GE.

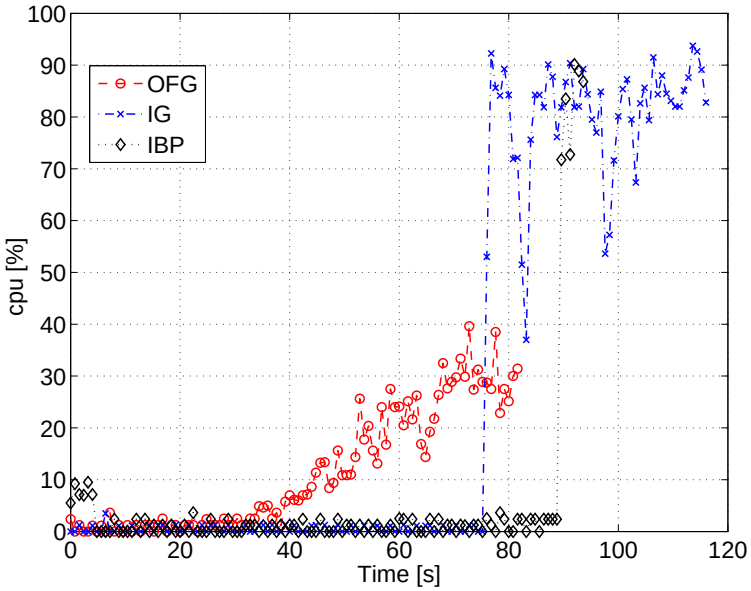


Figure 2.5: CPU usage (%) as a function of the time for 1Mbps transmission.

Chapter 3

Partial Decoding of Rateless Codes

3.1 Introduction

Rateless codes are designed to decode all the input symbols when enough coded symbols have been received. On the other hand, a receiver may have to decode the stream when an insufficient number of symbols has been received: in this case, it is not possible to recover all the transmitted information. However, it is possible to recover some of the information contained in the received coded symbols: this process is called *partial decoding* (26). In partial decoding, given an insufficient number of coded symbols, the decoder attempts to decode the maximum number of input symbols (the number of recovered input symbols is termed the *intermediate performance* of the code). Clearly, improving the intermediate performance is of paramount importance, especially when rateless codes are used to deliver multimedia content that exhibits graceful degradation in the presence of partial decoding. Indeed rateless codes are beginning to be exploited in several multimedia applications, e.g. MBMS in 3GPP and peer-to-peer video streaming (4).

To date, the decoding algorithm used for partial decoding of rateless codes is Message Passing (MP) (26). However, the MP decoder is designed for the decoding of rateless codes when the required number of coded symbols has been received, not for partial decoding. Previous work on this topic studied the following problem: how can we create a rateless code with good intermediate performance using the MP decoder? We

reverse that question and provide an answer to the following: given a certain rateless code, what is its optimal intermediate performance? Is there an efficient algorithm achieving such optimal performance? In particular, we introduce the concept of an optimal partial decoding algorithm. We propose the Optimal Partial Decoder (OPD) that, to the best of our knowledge, is the first optimal partial decoding algorithm for rateless codes. We also analyze its decoding complexity.

3.2 Intermediate Performance of Rateless Codes

All decoding algorithms are designed to attempt a complete decoding of rateless codes(e.g. to decode when a certain number $\bar{n}$ of coded symbols have been received). This bound, called the *overhead* of the rateless code, depends on the decoding algorithm used (29). What happens when this lower bound is not reached? In this case, it is of interest to know the intermediate performance of a code, i.e. the number of input symbols that can be retrieved from the received coded symbols. We call *partial decoding* the decoding process when the number of received coded symbols n is not enough to permit a complete decoding. We are interested in defining a optimality criterion for partial decoding algorithms. We say that a partial decoding algorithm is *optimal* if it is able to retrive the maximum number of input symbols, i.e. to maximize the intermediate performance, for every n.

The study of the intermediate performance of a rateless code can be viewed as the study of the linear space generated by the equations linked to the received coded symbols. Indeed, if we call E_n the linear space generated by the $n < \bar{n}$ received equations, the partial decoding problem can be recast as the search of degree 1 equations in E_n: if we call u_i the equation with a unique 1 in position i, an input symbol x_i can be recovered iff $u_i \in E_n$. A partial decoding algorithm can be seen as an algorithm that performs this search. Hence we can say that a partial decoding algorithm is optimal if, when given a linear space E_n, it is able to find all the degree 1 equations in E_n. I.e. it must be able to recover the maximum number of input symbols from an insufficient set of coded symbols, for every n.

The intermediate performance of rateless codes was initially studied in (17). In this crucinal paper, the author found an upper bound for the fraction of input symbols that can be recovered for any rateless code using the MP decoder. As shown in Fig. 3.1,

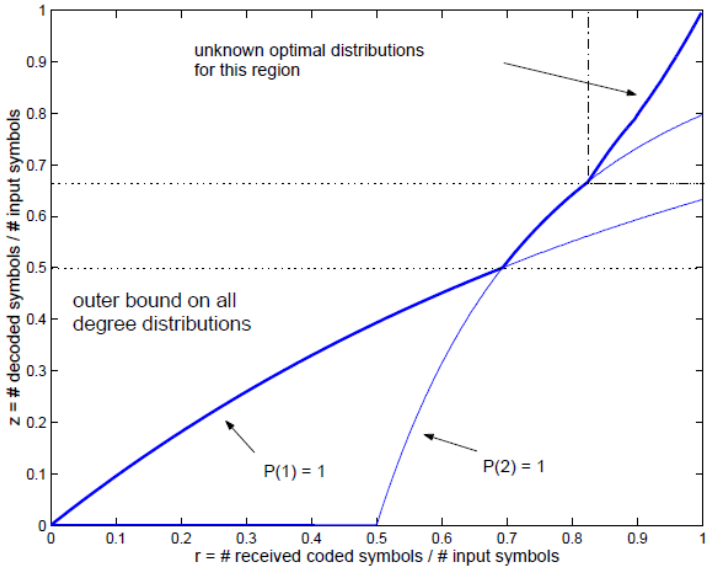


Figure 3.1: asymptotic fraction z of recoverable inputs as a function of the normalized number r of received packets (form (17))

The author divides the space of the percentage of received symbols into three regions. The asymptotic degree distributions that yield the better intermediate decoding performance are found in the first and second. The optimal degree distribution for the third region is unknown, but an upper bound is presented. However, the proposed degree distributions are asymptotic, so they are not always usable in a practical scenario. For example, the best degree distribution for the second region is $P(2) = 1$ (i.e. all the equations must have degree 2), but is not possible to decode this code. The xor of two even degree equations is again an even degree equation: it is not possible to obtain odd-degree equations from a set of even-degree equations, therefore it is not possible to obtain degree-one equations from such a set.

To solve this problem in (27) the authors use Pareto optimization to find practical optimal degree distribution. Weighted distributions are also proposed to obtain codes with good intermediate performance in all regions.

In (12) Growth Codes are presented: these codes, originally developed to maximize the data persistence (i.e. to maximize the probability of retrieving data) in sensor networks, are designed to optimize the intermediate performance of the MP decoder in a point-to-point scenario. A periodical feedback is required in these codes because

the degree distribution is tuned according to the number of symbols received by the receiver. The work in (28) studys the performance of a similar code, andshows that this feedback can be negligible.
All the works presented above seek to identify rateless codes with good intermediate performance when decoded with MP. Instead, we want to do something different: we want to find an algorithm able to maximize the intermediate performance of any rateless code.
In this chapter we present two optimal decoding algorithms for rateless codes and prove their optimality. The first one is very simple but computationally expensive. For this reason, it is used to derive the second one.

3.3 Naive Algorithm

An optimal partial decoding algorithm can be derived by classical Gaussian Elimination. If $n < \bar{n}$ coded symbols were received, it is possible to insert their linked equations in a $k \times k$ matrix M using GE. After the insertion, M will be an upper triangular matrix with up to k full rows. A row is full if it contains at least a 1, i.e. if it is not an all-zero row. Even if M is not full, we perform the back substitution of each row. If row i is full, we check the i-th column of M: when a 1 is found, it is canceled out by xoring that row with row i. At the end of the back substitution, M will have some rows of degree 1: these rows correspond with the decoded symbols. These are all the symbols that can be decoded starting from the n received symbols, i.e. GE with back substitution (GEB) is an optimal partial decoding algorithm. In this section we prove this claim.
We call e_i a generic k-symbols equation, r_j the equation corresponding to the j-th row of M and l_{e_i} the position of the leftmost 1 in the equation e_i.

Lemma 1. given n equations $\{e_1, \ldots, e_n\}$ such that $\forall i, j,\ l_{e_i} \neq l_{e_j}$ then the xor of the equations $e_1 \oplus \cdots \oplus e_n$ has the position of the leftmost 1 $l_{e_1 \oplus \cdots \oplus e_n} = min(l_{e_i})$.

Proof. If the positions of the leftmost 1 of the equations are all different, we can find an order of the equations. This can be done by sorting them by their l_{e_i}. Without loss of generality, let us assume that $l_{e_1} < l_{e_2} < \cdots < l_{e_n}$ so that $l_{e_1} = min(l_{e_i})$. If we xor the equations, the 1 of e_1 in position l_{e_i} cannot be canceled out by any other equation.

This happens because no equation has a 1 in that position. Otherwise that equation e_j should have $l_{e_1} = l_{e_j}$. For this motivation, $l_{e_1 \oplus \cdots \oplus e_n} = l_{e_1} = min(l_{e_i})$. □

Lemma 2. Denoting as u_i an equation with a unique 1 in position i, u_i can be obtained as a sum of a subset of n equations $e_1, \ldots, e_n$ such that $l_{e_1} < l_{e_2} < \cdots < l_{e_n}$ only if $\exists j$ such that $l_{e_j} = i$.

Proof. u_i has the leftmost 1 in position i, and following lemma 1, it is possible to obtain an equation with the leftmost 1 in position i only if $\exists j$ such that $l_{e_j} = i$. □

We denote as E_n the linear space generated by the equations $\{e_1, \ldots, e_n\}$ and we denote as $l^i_{e_j}$ the position of the i-th 1 of equation e_j.

Proposition 1. Given $n \leq k$ equations $e_1, \ldots, e_n$, $u_i \in E_n$ iff at the end of the GEB execution $r_i = u_i$.

Proof. If at the end of the GEB execution there exist some i such that $r_i = u_i$ then $u_i \in E_n$. We need to prove that GEB finds all u_i, i.e. that the procedure is optimal. Let l exists such that $u_l \in E_n$; we have to prove that, at the end of the procedure, $r_l = u_l$. At the end of the GE, some of the rows of M will be full, and all the full rows have a different position for the leftmost 1: hence it is possible to apply Lemma 2, stating that since $u_l \in E_n$ then the l-th row must be full. If $r_l = u_l$, the proposition is proven. Otherwise, r_l is to be xored with the other rows to cancel out its 1s. Let us find the position $p = l^2_{r_l}$ of the second 1 of r_l: following Lemma 1, it can be canceled without adding a 1 on its left only if r_p is not empty. Hence, we have to prove that r_p is not empty: indeed, since $u_l \in E_n$ and the rows of M form a basis for E_n then u_l can be obtained as a linear combination (i.e. xor) of the rows of M. Following Lemma 2, r_l must be used in the linear combination, so $u_l = r_l + q$, where q is obtained as a linear combination of the remaining rows. Obviously, $l_q = p$ and a row with a leftmost 1 in p exists for Lemma 1. Therefore, r_p is not empty. Row r_p can then be xored to r_l, and the result is stored as the new r_l. If $r_l = u_l$ the proof is done. Otherwise, the procedure can be repeated with the new $l^2_{r_l}$ until $r_l = u_l$. □

Proposition 1 proves that GEB is an optimal partial decoding algorithm. However, GEB needs to perform a Gaussian Elimination, which is a high-complexity algorithm, for each partial decoding attempt. For this reason, in the next section we propose an incremental optimal algorithm for the partial decoding of rateless codes that is based on GEB but has a lower decoding complexity.

3.4 An Incremental Algorithm

Algorithm 2 Optimal Partial Decoder (OPD)

```
Initialize the k × k-matrix M to 0
Initialize the k-vector y to 0
Initialize EmptyRows = k
while EmptyRows > 0 do
  receive a k-vector e and encoded symbol c
  for i from 1 to k do
    if e[i] = 1 and M[i][i] = 1 then
      e = XOR(e,M[i][·])
      c = XOR(c,y[i])
    end if
  end for
  find position l of leftmost 1 in e
  if l ≤ k then
    M[l][·] = e
    y[l] = c
    EmptyRows = EmptyRows − 1
    for i from 1 to l − 1 do
      if M[i][l] = 1 then
        M[i][·] = XOR(M[i][·],M[l][·])
        y[i] = XOR(y[i],y[l])
      end if
    end for
  else
    delete e, c
  end if
end while
```

The proposed Optimal Partial Decoder (OPD) is presented using pseudo code in Algorithm 2. The OPD is an *incremental* decoding algorithm, which is an algorithm in which the computational effort of the decoding process is distributed on all symbol receptions starting from the very first one. Initially, an all-zero $k \times k$ matrix M is created. According to this iterative procedure, when a new coded symbol c is received, the corresponding equation e is inserted in M. The position l of the leftmost 1 of e

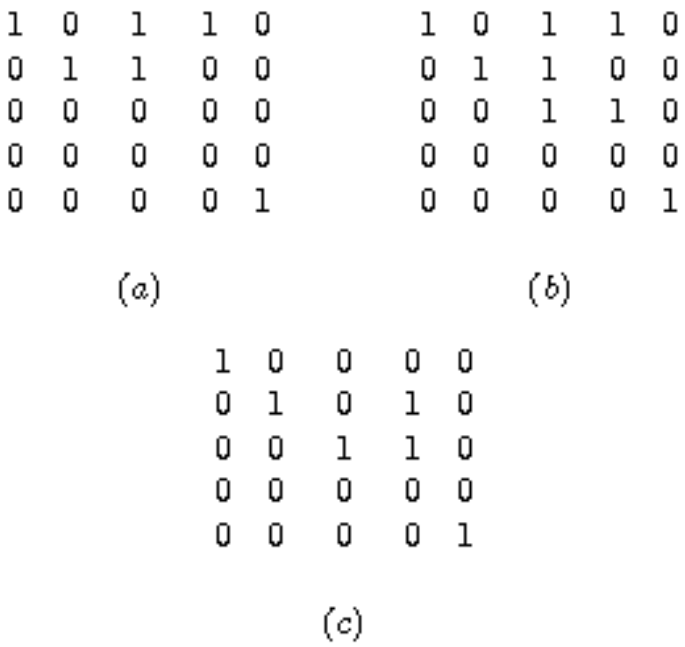


Figure 3.2: The evolution of matrix M during an OPD run

is found; if the l-th row r_l of M is full, the row and the equation are xored, obtaining $e = e \oplus r_l$. The corresponding coded symbols are also xored, resulting in $c = c \oplus y_l$ where y_l is the coded symbol of the l-row. Now the decoder finds the next 1 of e starting from l, and the equation and the corresponding row of the matrix are xored again, if full. The process is iterated until the end of the equation is reached. At this point, the new position l of the leftmost 1 of the equation is found and the equation is stored in the l-th row of the matrix (which was originally empty), i.e., $r_l = e$, and $y_l = c$. In the opposite case the equation gets discarded and OPD waits for the next received symbol. After the row insertion, a back substitution is performed: the l-th column of the matrix is checked, and when a 1 is found in row i, the l-th row is xored to row i, i.e. $r_i = r_i \oplus r_l$ and $y_i = y_i \oplus y_l$. After the back substitution, the algorithm waits for the next received symbol. The algorithm ends when all the rows of M are full.

As an example, let us assume that the receiver had the matrix in Fig. 3.2(a) and received the equation $e = 11101$. The equation has a leftmost 1 in the first position, so the decoder has to check if the first row is full. Since the first row is full, it is xored to

e obtaining $e = 11101 \oplus 10110 = 01011$. The next 1 is in the second position, and the second row of the matrix is full, therefore both row and equation are xored, obtaining $e = 01011 \oplus 01100 = 00111$. Now the third and the fourth rows of the matrix are empty, thus the equation does not change. The fifth row is full, and after xoring it to the equation we obtain $e = 00111 \oplus 00001 = 00110$. This equation has to be stored in the third row, as shown in Fig. 3.2(b), and the back-substitution step is performed: the first and the second rows have a 1 in the third column, ergo they will be xored to the third row obtaining the matrix in Fig. 3.2(c). Note that the first row now has degree one, so the first input symbol was decoded. Also note that if the i-th row of the matrix is full, then the i-th column has a unique 1 in the i-th row.

Proposition 2. OPD performs an optimal partial decoding of rateless codes.

Proof. We want to prove that OPD is an incremental version of the GEB algorithm; indeed, the insertion step of the algorithm is an incremental version of GE, in which GE is performed at each coded symbol arrival. When the received equation is inserted in the matrix, M is upper triangular. Moreover, the insertion step partially performs the back substitution procedure by xoring the equations among them canceling out some 1s. The rows of the matrix (the previously received equations) are used to cancel out the 1s of the new equation (the new row of the matrix) during the insertion, but the new equation is not used to cancel out the 1s of the previously received equations. The back substitution procedure is then completed by the second part of the algorithm, where the new equation (the new row) is used to cancel out 1s from the previously received equations (the old rows), as stated by the GEB algorithm. In this way, GEB is performed on the fly at each coded symbol reception. The optimality of the algorithm can also be noted by the fact that, at the end of back substitution phase, if the row i has a 1 in column $j > i$; then that 1 cannot be canceled using the row j because row j is empty. As previously discussed, this happens because, as already discussed, the column j has a unique 1 in the row j if row j is full. This means that at this point it is impossible to perform the GEB algorithm again using the rows of the matrix. Consequently, all the degree 1 equations are known and stored in their proper rows. □

3.5 Performance Evaluation of OPD with LT Codes

The performance of the proposed OPD algorithm in decoding LT codes is presented in this section. All the evaluations of the OPD will be performed using the values

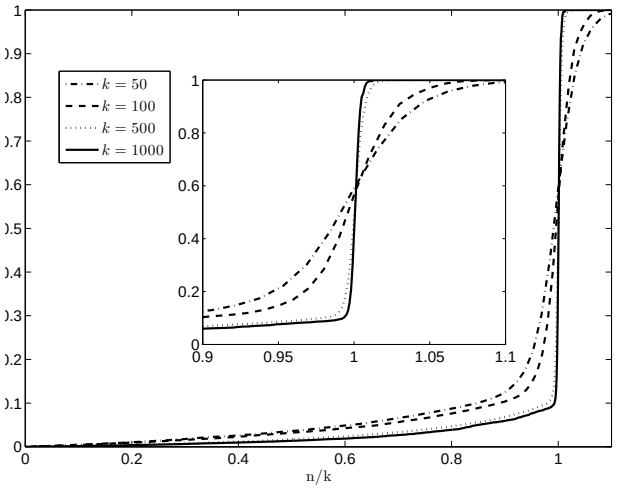


Figure 3.3: Partial decoding performance of LT codes.

$c = 0.05$ and $\delta = 0.01$ for the RSD distribution. As previously mentioned, we are not interested in comparing the partial decoding performance of MP and OPD algorithms. This would be an unfair comparison because, as proved in (29), a GE-based algorithm (such as OPD) has a lower overhead than MP algorithm. Instead, we wish to focus our attention on the effective intermediate performance of LT codes, using an algorithm that is able to reach the theoretical limit. In Fig. 3.3 the percentage of recovered input symbols is computed as a function of the percentage of received coded symbols. Both percentages are computed with respect to the total number of input symbols k. Recalling that OPD achieves optimal partial decoding, the only way to further improve the percentage of retrieved input symbols is to change the degree distribution, e.g. to use a different rateless code. In particular, note that the *all or nothing* behavior of LT codes; we can observe that the decoding transition gets sharper for larger block sizes. Finally, it is worth noting that when k coded symbols are received more than 50% of the input symbols can be retrieved independently on the value of k. In Fig. 3.4 the number of operations required by OPD to retrieve all the input symbols, computed with respect to the number of input symbols k, is shown. An operation is a row xor or a row swap. OPD complexity is compared to that of a single GE run, computed when k linear independent coded symbols are received. Since GEB needs a GE run per

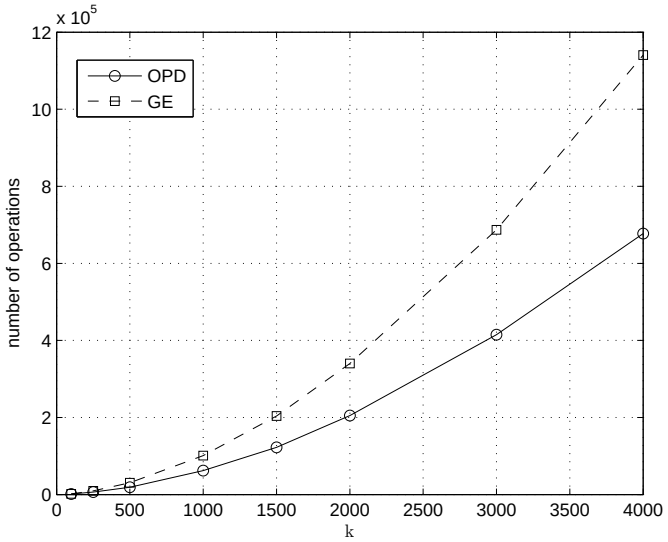


Figure 3.4: Algorithm complexity vs. k.

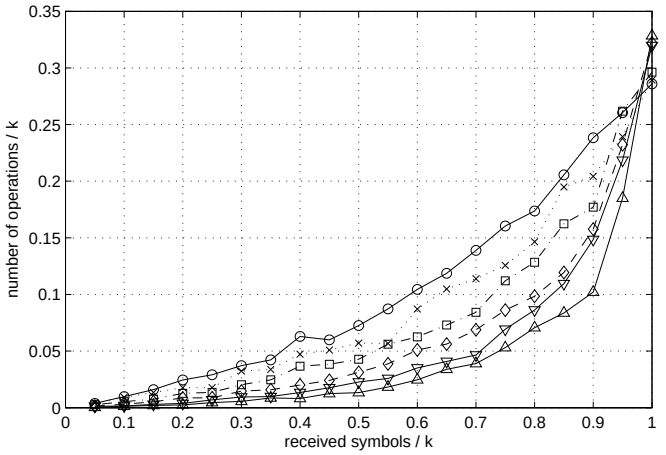


Figure 3.5: OPD complexity distribution per symbol reception.

received symbol, OPD is ultimately far less complex than GEB.

Another interesting feature of OPD is that it is an incremental algorithm, therefore the decoding complexity is spread over all the coded symbols received. In Fig. 3.5 the percentage of the operations, computed with respect to k, computed at each coded symbol arrival is shown with respect to the percentage of received coded symbols. This percentage can be viewed as the number of operations needed to insert and back substitute new coded symbols in M computed with respect to the the number of input symbols k. In Fig. 3.5 it can be noted that for each coded symbol reception less than $k/3$ operations are needed. Since the coded symbol reception time can be far larger than the time the CPU takes for an operation, it is possible to spread the decoding process along all the time used to collect the coded symbols. It is theoretically possible that, at each coded symbol arrival, all the previous received coded symbols were already processed and inserted in the matrix; this means that, at each symbol arrival, the OPD algorithm has already recovered the maximum number of input symbols.

Chapter 4

Rateless Codes and Network Coding

4.1 Network Coding Overview

Network coding (NC), initially proposed in (83), is a technique where the nodes of a network, instead of simply relaying the received packets, combine several packets before transmission in order to reduce the transmission cost. The essential concept is best explained with an example. Source S has access to packets b_1 and b_2 and has to send them to two sinks D_1 and D_2. The network is the so-called *butterfly network* shown in Fig. 4.1. All links have a capacity of one bit per unit time. The network problem can not be satisfied directly but only by forwarding the packets at the intermediate packet nodes. On the contrary, allowing the intermediate nodes to combine the packets can solve the problem, as shown in figure.

The main problem of NC is that the structure of the network must be known (and should belong to a certain family of topologies). To address this problem, the authors in (31) propose to combine the packets randomly. This strategy is called Random Network Coding (RNC). In fact, the sender creates a new packet via a linear combination, in GF(q), of its input symbols, where q is a sufficiently large integer. The other nodes in the network create new encoded packets through a linear combination in GF(q) of the previously received packets. However, in the paper it is proven that q should be larger than the number of the peers in the network. That makes this strategy hardly applicable in a real scenario, due its computational complexity, as stated in (32). In

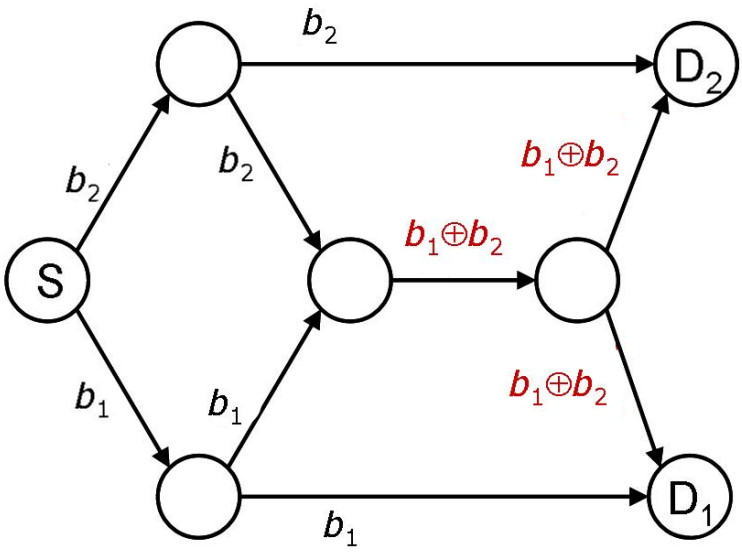


Figure 4.1: The Butterfly network

fact, in the case of $q = 2$ (and indirectly for $q > 2$), the RNC technique uses Dense Codes to create the packets to be forwarded by the peers. Dense Codes maintain a lower overhead, but conversely impose a high decoding complexity. Use of rateless codes in their place can solve this problem. In chapter 6 we will show a possible application of this idea using LT codes. However, in the following section we will prove that, whatever the initial degree distribution used to create the packets, the limiting distribution of the degree of the equations linked to the packets is the degree distribution of the Dense Codes.

4.2 Limit Degree Distribution

We want to study the expected degree distribution of a packet generated by the xor of a packet created following a certain degree distribution. We model this problem as follows. Our network is a complete binary tree of height h, as shown in Fig. 4.2. The transmission is directed from the bottom to the top of the tree, i.e. from the children to their parent. Every parent receives two packets, one from each child, and creates a new packet by xoring them. The leaves encode packets according to a certain degree distribution Ω^0, i.e. create packets of degree d with a probability of Ω^0_d. We call Ω^j the

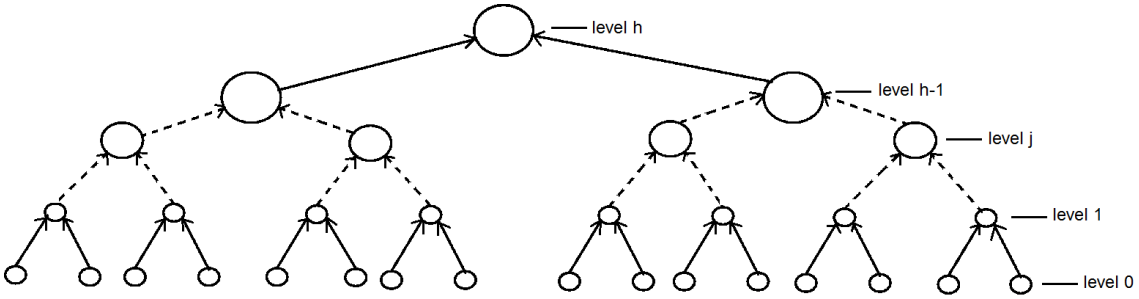


Figure 4.2: Network model

degree distribution of the packets in the system at level j, i.e. the degree distribution of a packet generated by a node of level j. This means that Ω^0 is the initial degree distribution of the packets. The probability Ω_i^j can be seen as the probability that the sum of two strings randomly chosen at level $j-1$ is i, therefore

$$\Omega_i^j = s_k(d_1, d_2, i) \quad \forall\, d_1, d_2, \tag{4.1}$$

where $s_k(d_1, d_2, d_F)$ is the probability that the sum of two randomly chosen strings of length k of degree d_1 and d_2 is a string of degree d_F. Using the law of total probability the equation becomes

$$\begin{aligned}\Omega_i^j &= \sum_{d_1=0}^{k}\sum_{d_2=0}^{k} s_k(D_1, D_2, i/D_1 = d_1, D_2 = d_2)\mathbb{P}(D_1 = d1)\mathbb{P}(D_2 = d_2) = \\ &= \sum_{d_1=0}^{k}\sum_{d_2=0}^{k} s_k(D_1, D_2, i/D_1 = d1, D_2 = d_2)\Omega_{d_1}^{j-1}\Omega_{d_2}^{j-1}.\end{aligned}$$

Now we focus our attention on $s_k(d_1, d_2, d_F)$. This probability can be calculated by checking the number of the common ones of the two initial strings. In fact, if X is the random variable that counts the number of positions in which both the strings have a one, it is possible to show that X follows the Hypergeometric Distribution, $X \sim \mathcal{H}(k, d_1, d_2)$. On the other hand, $d_F = d_1 + d_2 - 2X$, hence

$$\begin{aligned}s_k(d_1, d_2, d_F) &= s_k(d_1, d_2, d_1 + d_2 - 2X) = \\ &= \mathbb{P}(2X = d_1 + d_2 - d_F) = \\ &= \mathbb{P}(X = \tfrac{d_1+d_2-d_F}{2}) = \\ &= \frac{\binom{d_1}{\frac{d_1+d_2-d_F}{2}}\binom{k-d_1}{d_2-\frac{d_1+d_2-d_F}{2}}}{\binom{k}{d_2}}.\end{aligned}$$

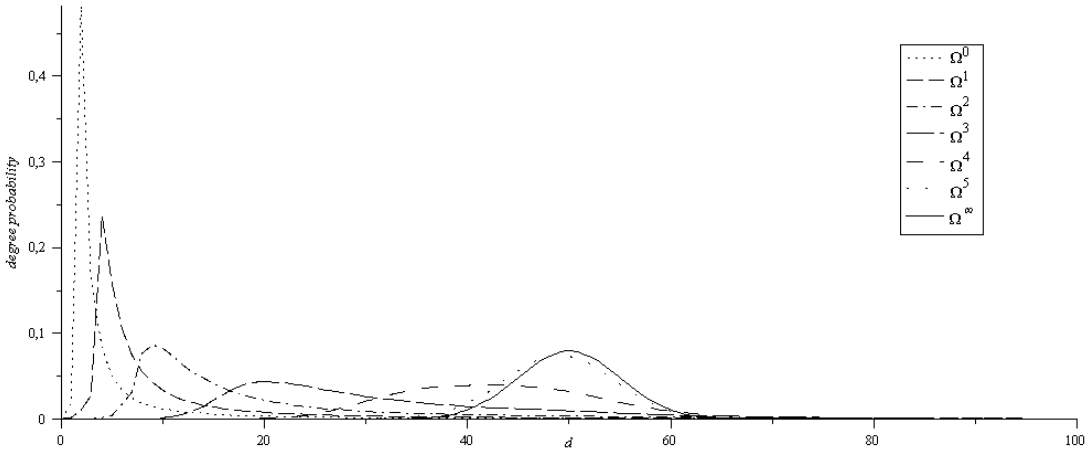


Figure 4.3: Evolution of the degree distribution with $\Omega^0 = RSD(0.1, 0.1)$ and $k = 100$

We recall that $\binom{k}{r} = 0$ when $r \notin \mathbb{N}$.
For $h \to \infty$, the limiting distributions Ω^∞ of the system depends on Ω^0; solving the equations 4.1, the limiting degree distribution converges to one of the following stable points, depending on the initial degree distribution Ω^0:

- If $\Omega^0_0 = 1$ or $\Omega^0_k = 1$ (i.e. the packets have always degree 0 or k) then Ω^∞ does not exist.
- If $\Omega^0_{2i+1} = 0$ (i.e. the packets have always even degree) or $\Omega^0_i = 1$ for $i = 1 \dots k$ (i.e. the degree is constant) then

$$\Omega^\infty_i = \begin{cases} \frac{\binom{n}{i}}{2^{n-1}} & \text{if } i \text{ is even} \\ 0 & \text{if } i \text{ is odd} \end{cases}$$

- Otherwise

$$\Omega^\infty_i = \frac{\binom{n}{i}}{2^{n-1}}.$$

In Fig. 4.3 the evolution of the degree distribution in the case $\Omega^0 = RSD$ is shown. It is of interest to note that the convergence to the limit distribution is rapid. The expected degree distribution of the system, excluding the limiting cases $\Omega^0_0 = 1$ and $\Omega^0_k = 1$, can be calculated as the average of the limiting distribution. Both the limiting distributions presented have $\mathbb{E}(\Omega^\infty) = \frac{k}{2}$, making the choice of the initial degree distribution virtually useless. In practice, whatever the initial degree distribution, the system converges to

the degree distribution of Dense Codes. Hence the encoding code will become a DC. However, as seen previously, the use of Dense Codes, even in a low-order Galois Field, makes RNC practically unusable, due the large complexity of the operations needed to decode packets. Therefore, it is evident that a higher control of the creation of the packets by node, and in particular their degree, is needed in order to prevent increasing of the complexity of the system. However, the RNC criterion, which gives excellent results in maintaining low overhead, should be maintained. These two conditions, degree control and use of RNC, would seem incompatibles. On the contrary, we prove that it is partially possible to match these two conditions. In our proposal, the degree control problem is solved through a new family of rateless codes, that we called *Band Codes*. These codes permit one to work in an RNC-like system while maintaining low overhead and high control on the degree distributions of the transmitted packets.

4.3 Band Codes

Band Codes (BC) share their primary concept with that of with Windowed Codes (WC). The idea is to create encoded packets by xoring only a subset of the set of the k blocks the message x is divided into. This subset is chosen to be the set of a certain number W of contiguous blocks. In fact, the encoder initially chooses the position m of the first block of the window; how m is chosen will be explained later. Then the encoder creates the encoded packet by xoring some of the W blocks in the window. Each block is xored with a probability of $\frac{1}{2}$. Band Codes can be decoded using Gaussian Elimination or, as we will see later, the OFG decoding algorithm, maintaining an interesting property.

4.3.1 Encoding

A Band Code, $BC(k, W)$, has two parameters: k, that is the number of input symbols, and W, the dimension of the window (hence $W \leq k$). In fact, to create a new encoded packet y_n the encoder chooses an element randomly $f^n = \{f_1^n, \ldots, f_W^n\}$ of $GF(2)^W$ and an integer $m \in [1, k - W + 1]$ with a certain probability. If the message x is divided into k blocks $\{x_1, \ldots, x_k\}$ then

$$y_n = \sum_{i=1}^{W} f_i^n \cdot x_{m+i-1}.$$

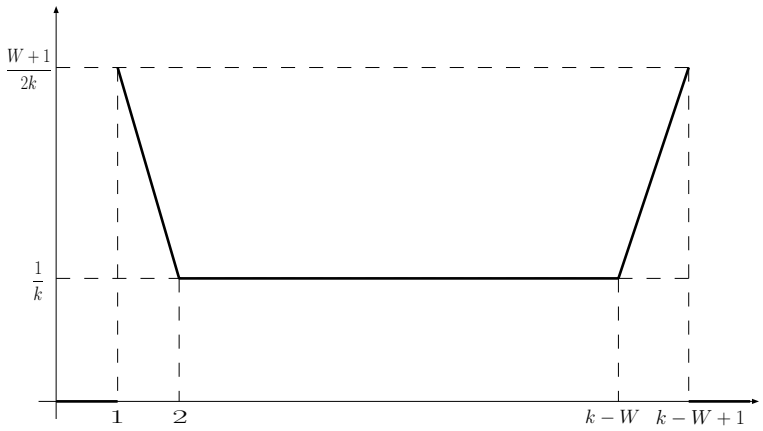


Figure 4.4: The horn probability density function

In fact, the encoding vector e_n of y_n is generated by f^n and m and has zeros in all the positions axcept in the window; it is called a *band vector*. The band vector is the main difference between Windowed Codes and Band Codes. In Windowed Codes, m is choosen randomly in $[1, N]$ and if $m_n > k - W + 1$ the window wraps. On the contrary, in Band Codes the window never wraps; if the window does not wrap, however, the probability that a block is used in the creation of an encoded packet is not uniform. This should imply that the overhead of the code is too high because a decoder has to wait to receive the low probability blocks. To overcome this problem, m is chosen following a particular distribution, called *horn distribution* HD:

$$HD(m) = \begin{cases} \frac{W+1}{2K} & \text{if } m = 1 \text{ or } m = k - W + 1 \\ \frac{1}{k} & \text{if } 1 < m < k - W + 1 \\ 0 & \text{otherwise} \end{cases}$$

The horn distribution is plotted in Fig. 4.4. Later we will see that, by using the HD to choose m, we ascertain that the overhead of the code can be controlled and keep very low.

4.3.2 Decoding

We created BC despite the existence of WC because of the particular shape that the decoding matrix maintains during the decoding process.
Band Codes are Rateless Codes, thus they can be decoded using the classical decoding algorithms used to decode rateless codes. As shown in (29), however, the overhead of the code also depends on the decoding algorithm. Due to the low expected number

of degree one packets, the overhead of Band Codes using the BP algorithm will be very high. Hence the decoding algorithm used for BC should be a GE-like algorithm. However, if the OFG algorithm is used, the decoding matrix G maintains an useful property:

Proposition 3. If the OFG decoding algorithm is used to decode a Band Code BC(k, W) then the decoding matrix G is a band matrix with a lower bandwidth equal to zero and an upper bandwidth equal to $W - 1$ for each packet arrival.

Proof. The proof is made by mathematical induction on the number of received packets.
Basis: at the first packet arrival G is empty, therefore the packet will be inserted in a certain row r of the matrix. After the insertion, the matrix has a unique non-zero row r, that is equal to the received packet. That packet was generated using a BC(k, W), proving the proposition.
Inductive step: we suppose that the proposition holds for the first $k-1$ received packets, i.e. G is a band matrix after the reception of $k - 1$ packets. The packet p is received, and OFG algorithm attempts to insert it into the matrix. If l is the position of the leftmost 1 of p, we have two cases:

- If $G[l][l] = 0$ (i.e. the l-th row of G is empty) then p can be immediately inserted in G. After the insertion, G is a band matrix for the same reason shown in the Basis step.

- If $G[l][l] = 1$ (i.e. the l-th row of G is already full) then a xor between the row $G[l]$ an p has to be performed (after a potential swap between $G[l]$ and p, after which G is still a band matrix for the precedent assertion). If we call $p' = p \oplus G[l]$, we have that if p' is still a band vector then the proposition holds.
  The position l' of the leftmost 1 of p' is greater than l; $G[l]$ and p have the leftmost 1 in the same position l, which will be cancelled out by the xor operation. If we call r and r_G the positions of the rightmost 1 of p and $G[l]$ respectively, because of the xor operation we have that the inequality $p' \leq max\{r, r_G\}$ holds for the position r' of the rightmost 1 of p'. This means that $r' - l' \leq r - l \leq W$, ergo p' is still a band vector.

□

This proposition will be very useful for the creation of a practical RNC system in chapter 7.

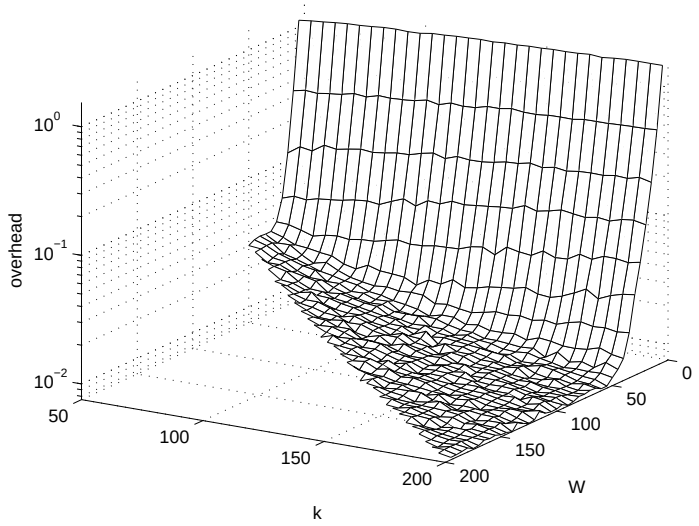


Figure 4.5: Overhead of Band Codes in end-to-end transmission

4.3.3 Code Analysis

In this subsection we want to analyze the performance of Band Codes in an end-to-end scenario, where the packets are transmitted to the receiver directly from the sender; the case of the use of BC in a RNC scenario will be expamined further in chapter 7, where we will use BC to create a video streaming protocol on a network that permits RNC .

Band Codes belong to the family of rateless codes, therefore the overhead is one of the performance indeces to be studied. In the previous sections we mention that the Horn distribution assures a low overhead. In fact, the overhead depends on the dimension of the window W. If $W = k$, BC collapse to DC, i.e. $BC(W, k) \rightarrow DC(k)$ for $W \rightarrow k$. The more the window decreases, the more the overhead augments. However, as shown in Fig. 4.5 the overhead still remains very low even if $W \ll k$. Due to a particular property of random matrices presented in (35), there is a high probability that a binary random $k \times k$ matrix composed of rows of average degree d is full-rank until $d > 2 \log k$. However, in this instance, the random matrix is not a band matrix. In the case of random band matrices of band W, the authors in (11) propose the conjecture that if $W > 2\sqrt{k}$, then the full-rank property still holds. This conjecture is confirmed by the low overhead of BC in the case of $W > 2\sqrt{k}$; however, further analysis needs to be

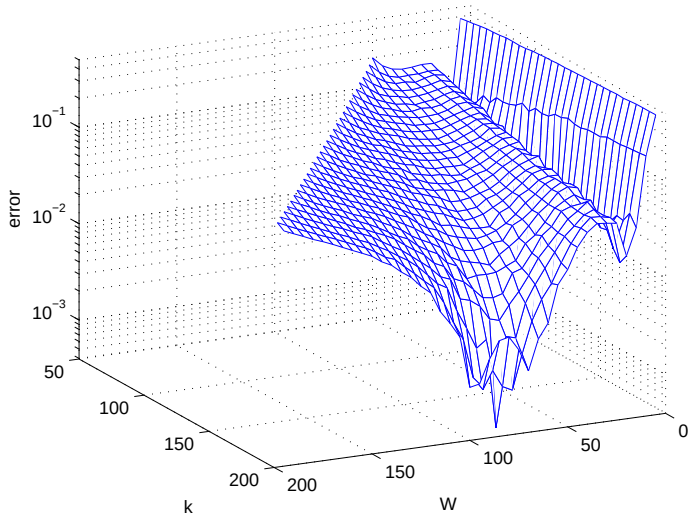


Figure 4.6: Relative error of the proposed model in the calculation of the complexity of the decoding process

done.

From the computational complexity point of view, the number of operations needed to decode Band Codes also depends on W. The *decoding cost* C_D is defined as the average number of XOR operations performed by the OFG algorithm in order to decode. The OFG algorithm is composed of two distinct stages; for this reason, we need to separately model the computational cost of each stage. The cost of the first OFG stage, $C_{D,I}$, grows by one unit every time a received packet collides with a row of G during the execution of the OFG. Collisions take place with a probability that depends on the number of rows of G, i.e. as G fills up, collisions are more likely. Given that, after receiving i packets, G has approximately a rank equal to i, the maximum number of collisions that the decoding of a packet can generate is i. In turn, the probability that the algorithm iterates depends on the average degree of the packet and of the rows of G. In the case of Band Codes , these turn out to have the same value of $d = \frac{W}{2}$. If we suppose that a collision happens with probability $\frac{d}{k}$ and that $\epsilon \simeq 0$, we are able to calculate an upper bound to the expected cost of the first stage of the OFG algorithm as:

$$C_{D,I} = \sum_{i=0}^{k-1} i\frac{d}{k} = \frac{d(k-1)}{2} \simeq \frac{dk}{2} = \frac{W}{4k}.$$

The cost of the second OFG stage $C_{D,II}$ is equal to the number of 1s in G, i.e. it depends on the average degree of the rows of G at the moment of diagonalization. As already seen, the OFG algorithm ensures that G is a band matrix, hence we assume that the expected degree of a row is $\frac{W}{2}$. Thus the computational cost of the second OFG stage is:

$$C_{D,II} = \frac{W-1}{2}k \simeq \frac{Wk}{4}.$$

Finally, we write that the total decoding cost is

$$C_D = C_{D,I} + C_{D,II} \simeq \frac{WN}{2}. \tag{4.2}$$

Fig. 4.6 shows the relative error committed by the model of Eq. 4.2 in the calculation of the complexity of the decoding process. It is possible to note that our model is very close to the effective complexity, in particular for $W \simeq 2\sqrt{k}$. The large error for $W \to 0$ is due to the larger overhead. Fig. 4.5 indicates that the hypothesis $\epsilon \simeq 0$ is no longer valid in this case. On the other hand, the good adherence of the error model for $W \simeq 2\sqrt{k}$ enables evaluation of the complexity of the decoding algorithm as $O(k\sqrt{k})$. The low overhead and the controlled computational complexity suggest $W \simeq 2\sqrt{k}$ be imposed as a parameter for Band Codes.

Chapter 5

Rateless Regenerating Codes

5.1 Introduction

In recent years, erasure codes have been used in distributed storage systems to increase the reliability. The original information is encoded in a number of fragments such that every subset of these fragments (of size k) is sufficient to retrieve the initial information. Afterwards, these fragments are distributed to the nodes of the system, usually one per node. Because of the use of coding schemes, however, a new problem arises: when a node fails or the overall reliability must be increased, a new encoded fragment must be created. In this case, the use of conventional erasure codes is an obstacle: in fact, to create a new encoded fragment, it is mandatory to use the original information. Hence this information must be retrieved. As a consequence, the maintenance cost of the system, in terms of bandwidth usage and computational complexity, could overwhelm the advantages of increased reliability.

Regenerating codes were created to face the problem of error repair, i.e. to recreate the fragments lost in a node failure. A key feature of these codes is that they are able to minimize the bandwidth necessary to repair a failure. After the seminal paper of Dimakis et al. (23), many regenerating codes schemes were proposed. When a node fails, all the remaining nodes send a part of the information they store to a newcomer in order to recreate the lost fragment. In all the schemes, however, the number of nodes involved in the process is fixed and larger than k. Usually they solve only the case of a unique failure. Moreover, these schemes do not solve the reliability increase problem: when a new fragment has to be added to the system, all the fragments must

be recalculated.
We propose a scheme that solves both these problems using fountain codes.

5.1.1 Regenerating Codes

The model of the distributed storage problem is as follows. A message x of size $\mathcal{M}$ has to be stored across n storage nodes. The owner of the message, called *source*, encodes the message through an erasure code E into n fragments of size α such that any k out of the n fragments are able to retrieve the initial message - termed *reconstruction property* - and sends each fragment to a storage node. When a node fails after leaving the network, a newcomer joins the system; d out of the remaining $n-1$ nodes, with $d \leq k$ fixed a priori, send a fragment of size $\beta \leq \alpha$ to the newcomer. That newcomer can recreate the lost fragment of size α using the received fragments.An important parameter of a regenerating code is its repair bandwidth. The repair bandwidth γ is defined as the total size of the information flows in the network to repair or recreate an encoded fragment. If d nodes send information of size β to the newcomer, $\gamma = d \cdot \beta$. Obviously, the system is efficient if $\gamma \leq \mathcal{M}$; otherwise it would be better to decode the information in order to create a new encoded fragment.
In (23) the authors prove the existence of an optimal tradeoff curve between the size of the stored fragment α and the repair bandwidth γ; in particular, the authors focus their attention on the two extremal points of the curve, which correspond to minimum-storage regenerating (MSR) codes and minimum-bandwidth regenerating (MBR) codes. In (36) a survey of the topic is presented, including the explicit construction of many regenerating codes.
Nevertheless, all the results are obtained in the case of $d \geq k$. On the contrary, the authors say it would be inevitable to download (and retrieve) all of the message in order to recover the lost fragment. In this chapter we also show that, relaxing the reconstruction property through Fountain Codes (10), it is possible to disprove this conclusion.

5.1.2 Rateless Codes: a Remark

A message of size $\mathcal{M}$ is encoded into a theoretically infinite stream of fragments of size $\alpha = \frac{\mathcal{M}}{k}$, that are sent to the receiver through an erasure channel. A *relaxed reconstruction property* holds for Rateless Codes: any $k(1+\epsilon)$ fragments are sufficient

to retrieve the initial message with high probability, with $\epsilon \to 0$ for $k \to \infty$. In fact, the decoding of a Rateless Code can be performed when k linearly independent fragments are collected; these codes are designed such that any subset of $k(1+\epsilon)$ fragments contains k linearly independent fragments with high probability (that directly depends on ϵ). Another important property of Rateless Codes is the low encoding and decoding complexity due to the structure of these codes.
In (10) a survey of Rateless Codes is presented; after the paper by Luby (9), which introduces LT Codes, many Rateless Codes were created with various features. For our proposal, however, the most important feature, common to all Rateless Codes, is the relaxed reconstruction property, along with the possibility of creating new fragments on the fly.

5.2 Encoding

The encoding of a Rateless Regenerating Code (RRC) works as follows. The message is divided into $k < n$ parts $x = \{x_1, \ldots, x_k\}$ and encoded through a fountain code F in a stream of encoded fragments $c = \{c_1, \ldots, c_n\}$. Every encoded packet is obtained as the Xor of input fragments $c_i = \sum_{j=0}^{k} g_{i,j} x_j$, where the encoding vectors $g_{i,\cdot}$ depends on the fountain code chosen for the encoding. We call G the encoding matrix that contains the encoding vectors as rows, i.e. $G \cdot x = c$. We assume that the first k encoded fragments $c' = \{c_1, \ldots, c_k\}$ are linearly independent, i.e. that the first k rows of G are linearly independent; otherwise it is possible to permute the rows of G in that way. We call G' the submatrix of G obtained by the first k rows of G and G'' the remaining part of the matrix; obviously, $c' = G' \cdot x$. Because of the linear independence of the firsts k encoded fragments, it is possible to invert G' by solving the system, obtaining $x = (G')^{-1} \cdot c' = M \cdot c'$, with $M = (G')^{-1}$. We call the rows of G' *basis vectors*, and the corresponding fragments of c' *basis fragments*. Now each encoded fragment c_{k+l} can be obtained as a linear combination of the basis fragments of c' in such a way that:

$$\begin{aligned} c_{k+l} &= \sum_{i=1}^{k} g_{k+l,i} x_i = \sum_{i=1}^{k} g_{k+l,i} \left(\sum_{j=1}^{k} m_{i,j} c_j \right) = \\ &= \sum_{j=1}^{k} \left(\sum_{i=1}^{k} g_{k+l,i} m_{i,j} \right) c_j = \sum_{j=1}^{k} v_{l,j} c_j \end{aligned}$$

We call $v_{k+l,.}$ the *recoding vector* of the encoded fragment c_{k+l}, named as *recoded fragment*. The recoding vector of the basis fragment c_l, $l \leq k$, is a vector that has a unique 1 in position l. In general, the encoding vector $g_{l,.}$ can be transformed in the recoding vector $v_{l,.}$ using M as $v_{l,.} = g_{l,.} \cdot M$, hence $v_{l,.}$ is obtained Xoring the rows of M for which the encoding vector has a 1. If we call V the matrix formed by the recoding vectors as rows, obviously $V = G \cdot M$. The upper part of V is hence the identity matrix I_k.
After the encoding, the fragments are distributed among the n storage nodes, one for each node. The source stores the position of each fragment in the network.

5.3 Decoding

The decoding of a RRC can be performed in two ways: using either the encoding or the recoding vectors.
In fact, if the encoding vectors that form the matrix G, are stored along with the fragments, it is possible to retrieve the message x decoding the rateless code F through the usual decoding algorithm of that code. The source collects $k + \epsilon$ fragments and uses their encoding vector to decode the code normally.
As we will see in the next section, however,in order to repair the lost fragments, the recoding vectors have to be used. Hence matrices V and M have to be stored (at least by the source). In order to reduce the spatial complexity of the system, it is desirable to use these matrices also in the decoding phase instead of using G. G could hence be cancelled.
In fact, if the source is able to collect all the basis fragments, matrix M can be used to recover the initial message calculating $x = M \cdot c'$. However, it is not mandatory to collect directly the basis fragments. Given any subset of k linearly independent fragments (containing some recoded fragment or not), it is possible to recover all the basis fragments. Let us suppose that the source collected the k linearly independent fragments $\{c_{l_1}, \ldots, c_{l_k}\} = r$. We call R the matrix formed by their recoding vectors as rows; R is a submatrix of V. Following the definition of recoded vectors, the equation $R \cdot c' = r$ holds. Solving this system, it is possible to retrieve the basis fragments c'. Also in this case, for the relaxed reconstruction property, it is sufficient to collect $k + \epsilon$ fragments in order to assure that an invertible matrix R is found.

5.4 Errors repairing

Using RRC it is possible to compute a distributed encoding of new encoded fragments as well as the recovery of an encoded fragment without decoding the entire message.
When a newcomer joins the system and a new encoded fragment c_{n+1} must be created, the source generates the encoded vector $g_{n+1,.}$ through the fountain code F. Now the recoded vector $v_{n+1,.}$ is generated from the encoded vector as $v_{n+1,.} = g_{n+1,.} \cdot M$. The nodes that own a basis fragment involved in the encoding of that fragment (i.e. a basis fragment c_j such that $v_{n+1,j} = 1$) are asked to send their fragment to the newcomer. The newcomer receives the fragments and Xors them to obtain the new encoded fragment, then the received fragments are deleted.
When an encoded fragment is lost, it can be recovered in a similar manner. If a fragment c_l is lost, with $l > k$, it is possible to recreate it following the procedure of creation of a new encoded fragment. In this case, however, it is not necessary to re-create the same encoded fragment, and a new encoded fragment can be created instead. If the lost fragment c_l is a basis fragment, hence $l \leq k$, the lost fragment is crucial for the creation of new encoded fragments. Therefore it must be recreated. To recover c_l, a recoding vector $v_{k+s,.}$ is searched such that $a_{k+s,l} = 1$, i.e. an encoded fragment c_{k+s} obtained by an Xor that involves c_l. In fact, if $v_{k+s,l} = 1$, then:

$$c_{k+s} = \sum_{j=1}^{k} v_{k+s,j} c_j = c_l + \sum_{\substack{j=1 \\ j \neq l}}^{k} v_{k+s,j} c_j,$$

and inverting this equation, we obtain that

$$c_l = c_{k+f} + \sum_{\substack{j=1 \\ j \neq l}}^{k} a_{k+f,j} c_j.$$

To recover c_l it is possible to "cancel out" from c_{k+s} all the encoded fragments involved in its creation but c_l. All the basis fragments involved in the creation of c_{k+s} (but c_l, obviously) and c_{k+s} are sent to a newcomer that Xors them in order to recover c_l.

5.5 Repair Bandwidth

We want to calculate the repair bandwidth of the system. In the system we present, the nodes send all their information to the newcomer. This information is created

using a Fountain Code, therefore $\beta = \frac{\mathcal{M}}{k}$. The system is efficient if $d \leq k$, i.e. the efficiency depends on the number of nodes involved in the recoding process. For each encoded fragment, this number is given by the number of 1s in its recoding vector, that is the degree of the vector: for the fragment c_{k+l} the number of nodes involved is $d = \sum_{i=1}^{k} v_{k+l,i}$ (a similar property holds for the fragments in c'), therefore $d \leq k$. In particular, a recoding vector is obtained as a linear combination of rows of M, that is the inverse of the encoding matrix G of the code. As long as G is a sparse random matrix, M is a random matrix with high probability, where on the average each row contains $\frac{k}{2}$ ones. A recoding vector is then obtained as the xor of rows of a random matrix; therefore, recoding vectors are random vectors of expected degree $d = \frac{k}{2}$. This means that, in average, for RRC, the repairing bandwidth is

$$\gamma = d \cdot \beta = \frac{k}{2} \cdot \frac{\mathcal{M}}{k} = \frac{\mathcal{M}}{2} < \mathcal{M}.$$

5.6 Conclusions

We propose a general framework to create regenerating codes able to reconstruct the lost fragments under the bounds proposed in (23), i.e. contacting less than k storage nodes. Moreover, the proposed framework solves the reliability increase problem, unsolved for the classical regenerating codes. These results are obtained by relaxing the reconstruction property of regenerating codes, enabling the use of sub-optimal codes as rateless codes. The use of rateless codes, along with a novel point of view on the problem, permits one to create and repair fragments in a distributed fashion. The proposed Regenerating Rateless Codes also assure an unexpectedly low repair bandwidth.
From the repair bandwidth point of view, MSR codes (23) can achieve better performance with the same fragment size α; however, that result holds for $d > k$, while for $d \leq k$ the repair bandwidth should be larger. Moreover, MSR codes do not solve the reliability increase problem.

Part II

Application

Chapter 6

Practical Random Network Coding using Rateless Codes

6.1 Introduction

Several recent results pointed out that the use of coding techniques increase the efficiency of content distribution applications such as the reliable distribution of bulk data, the application level multicast, P2P streaming applications (1, 2, 3, 4) or efficient broadcasting in ad hoc wireless networks (5, 6), just to mention a few. Most of the cited results have been catalyzed by the seminal promises of network coding (8, 37), where nodes in the network are allowed to combine information packets instead of simply forwarding them. In particular, in Random Network Coding (RNC) (31) each peer transmits linear combinations of incoming packets, where the coefficients are chosen randomly over some finite field. The deployment of network coding at theapplication level, e.g., in the field of P2P file sharing or video streaming, has been limited primarily because of the added computational cost due to linear coding. Nowadays, such a complexity issue must be carefully reconsidered; indeed, rateless codes, designed for application level coding, is turning out to be a practical tool for efficient coded data dissemination. From the point of view of the added computational cost, rateless codes are required to perform simple xor operations among the original packets on the encoder side, and to solve a sparse linear system in a Galois field of order 2 on the decoder side.

In most of the deployed P2P applications, a peer concurrently downloads contents from

multiple peers and uploads towards multiple peers. Although this improves the bandwidth utilization and allows one to counteract network dynamics, content reconciliation policies are required. The potential advantage of coding, however, is the simplification of the content reconciliation problem, since every piece of coded information is equally useful regardless of the peer who has contributed it. If one uses coding, in principle, there is no limit to the number of uniquely encoded packets generated from the original set of packets, thus relaxing the content reconciliation issue. Therefore, a simpler approach can be adopted to propagate the information. Nonetheless, coding poses novel issues, as well. In particular the information flow has to be divided into coding blocks; the computational complexity needed to encode and decode such blocks could make this approach unfeasible in practice.

For these motivations, we propose a practical Random Network Coding strategy in GF(2) using rateless codes. Our goal is to design a more efficient exploitation of DF in principle for both fast propagation of the information and low communication overhead. This eliminates the transmission of an excessive amount of redundant coded packets. We use LT codes (9) to pre-encode the source information. In general, when a node that has not decoded yet has the possibility to send a packet, it sends a linear combination of the previous received packets, calculated in GF(2). We propose a low-complexity strategy to calculate such a linear combination. Moreover, we propose to throttle the initial bandwidth of the peers to decrease the number of duplicated packets. The result is that the information is spread over the network while maintaining a low number of duplicated packets and an acceptable complexity in encoding and decoding, even though we use a RNC-like strategy.

A huge literature is present in this field. In (1) an in depth analysis of the reconciliation issues in conjunction with packet encoding is shown. A set of reconciliation algorithms trading off accuracy and complexity is proposed. Ref. (1) designs a family of reconciliation techniques, which have also been tested in a real test-bed in (2), through which the peers participating in the overlay attempt to coordinate the content downloading by means of both original packet coding and recoding of already coded data.

The authors in (38) follow a complementary approach. They avoid the need for reconciliation based on the optimal design of a distributed rateless code, i.e., coded packets are guaranteed by construction to be independent and equally useful. Nevertheless the solution proposed in (38) is limited to the case of a single network topology with a

common relay node, that can be generalized only assuming perfect knowledge of the overlay connections. This latter assumption is clearly unfeasible in a real dynamic overlay. Moreover, the optimization algorithm complexity increases with the size and the number of the connections in the overlay. In (39) another optimal coding approach is proposed. This solution is based on re-encoding, where several coding stages are cascaded while moving from hop to hop. Besides being asymptotically optimal, recoding comes at a significant computational expense in intermediate nodes, and may lead to excessive coding overhead in a real scenario with several hops and limited code block length.

A simpler approach which is able to cancel the reconciliation phase is used in (4), where P2P streaming application adopting rateless coding and optimal peer selection is presented. In (4) the DF approach is applied on every peer-to-peer connection and peers are not allowed to propagate coded information before the complete decoding of a block. Therefore, at the expense of an additional delay, every peer waits for complete decoding of a block of original packets and then starts sending independent LT encoded packets. This strategy has low decoding and decoding complexity due the performance of LT codes, but the time needed to spread information in a network is large because the nodes have to wait to decode a whole block before starting to relay. The proposed push approach is coupled with an optimal overlay formation strategy aiming at constructing high quality streaming topologies so that end-to-end latencies are minimized.

In (31) the authors propose to combine packets to achieve better performance: this strategy is called Random Network Coding (RNC). In fact, the node that owns the information to be spread, called the *seeder*, creates a new packet by a linear combination, in GF(q), of its input symbols, where q is a sufficiently large integer. The other nodes in the network create new encoded packets by a linear combination in GF(q) of the previously received packets. The paper proves that q should be larger than the number of the peers in the network. That makes this strategy hardly applicable in a real scenario due its computational complexity, as stated in (40). To face this problem in (40), a peer waits to relay packets until it has received a certain number of packets. Using this parameter it is possible to decrease the information spreading time, even if the performance of RNC is still considered poor. In fact, the computational complexity of encoding and decoding in GF(q) increases with the increasing of q, and is still too costly.

The lesson we learn from (40) is that RNC is not feasible in a real scenario because all the encoding and decoding operations are made in a large-order Galois Field. Therefore, we propose strategies to increase the performance of RNC in low-order Galois Fields. We use LT codes as a source pre-code for two motivations: they have a low decoding complexity, and they are generated in GF(2), i.e. with the lowest possible value of q. Similarly, we use the OFG decoding algorithm to decode received packets because it permits one to decrease the time needed to create new packets thorough the combination of previously received packets (i.e. what in RNC is called encoding time). This is possible because, using OFG, the rows of the decoding matrix are random linear combinations of received packets; when a node needs a linear combination of packets to encode a new packet it can to use these rows rather than calculate a new linear combination. In addition, OFG yields a low overhead ϵ even if the degree distribution is not RSD, thus allowing one to exploit linear combinations of LT encoded packets within an RNC delivery approach.

The performance of RNC improves with the number of received packets: when a node has received few packets, the encoding process (i.e. to create new packets by a linear combination of the previously received packets) is less useful. In opposition to (40), which proposes to block the recoding process, we propose instead to limit the number of linear combinations injected by a node depending on the number of received coded packets.

6.2 System description

In this section we describe the behavior of the various peers in the network and the relaying strategies they can follow.

We consider a distributed application whose components have organized in a peer-to-peer overlay network $\mathcal{T}$. We make no hypothesis on how $\mathcal{T}$ is formed therefore multiple paths between pair of peers and cycles may be allowed. There is a single peer that holds valuable information for all the others (the original source). At startup all other peers are interested in retrieving the information and cooperate to obtain it. Every peer stores the coded packets that turn out to be useful in the buffer OB. By a useful coded packet we mean a received packet that is not linearly dependent on the previously received ones. Each peer is allowed to combine and forward packets from its buffer OB.

Peers are characterized by their upload (B_u) and download (B_d) bandwidth expressed in *bps*. In the sequel we also need to express the peer bandwidths in *pps* (packets per second) for a given packet size; in this case we denote the peer bandwidths as N_u and N_d.

6.2.1 State of the peers

Each peer in $\mathcal{T}$ can be in one of the following states:

- WAIT: the peer is waiting for the reception of the first coded packet. As soon as the peer receives the first packet, it changes its state to DECODER.
- OFF: the peer is not cooperating to obtain or distribute the data. This state is assumed when the peer and all its neighbors have already received and decoded all the k packets.
- SEEDER: the peer has already received and decoded the k information packets but some of its neighbors are still in the DECODER state. In this case, the peer generates new LT coded packets, saturating the upload towards its neighbors. As soon as all its neighbors have decoded the original information, the peer changes its state to OFF.
- DECODER: the peer has not received enough information data to decode the original information. The relaying strategy followed by peers in state DECODER will be discussed in Section 6.2.2. As soon as the peer receives enough packets to decode the information data, the peer signals such event to all its seeding nodes in order to stop them from pushing more coded packets and changes its state to SEEDER.

At the beginning, the source state is set to the SEEDER state while all the other peers are set to WAIT state. All peers in the SEEDER state encode the original data and send it to their neighbors in the DECODER state using the RSD. All peers in the DECODER state run the OFG decoding algorithm and progressively construct their generator matrix G, based on the generating equations of the received coded packets. At the same time, these peers insert only their *useful* packets in an output buffer OB from which packets are selected for relaying.

6.2.2 Protocol description

Throttling peers in DECODER state and combining packets before relaying to neighbors in DECODER state are effective strategies to reduce the amount of duplicated packets while retaining the capability of spreading received packets as soon as possible. This allows us to define and compare several strategies for the relaying of coded packets while in the DECODER state:

- Store and Recover (SR): a peer does not forward any of the received coded packets that are used to recover the k original blocks. This means that a peer starts to forward packets only when it switches to the SEEDER state. This is the strategy used in rStream (4).

- Relay (RE): at every transmission opportunity, a peer selects a packet in OB and forwards it. Such a packet is deleted from OB in order to relay it only once. The procedure is repeated until OB is empty or the upload capacity is saturated. This strategy is used in (41).

- Random Relay (RR): a peer at every transmission opportunity randomly draws from OB enough packets to fully use its upload capacity and sends them to its neighbors.

- Random Relay with Combinations (RRC): at every transmission opportunity the peer randomly draws from OB enough packets to fully use its upload capacity, it xors them with a randomly chosen row of the decoding matrix $\mathbf{G}$ and sends them to its neighbors. This amounts to combining the selected packet with a set of previously received packets at the cost of a single packet xor operation.

The aim of the RR and RRC strategies is to send as much information as possible: the high utilization of upload bandwidth reduces the information data spreading time. These strategies may be too aggressive, i.e. they could fill the network with too many duplicate packets. For this reason we consider two variants of previous strategies, namely TRR and TRRC, where the upload bandwidth of RR and RRC is *throttled*. In particular, at any transmission opportunity the number of relayed packets is limited to $\min(N(t)/\alpha, N_u)$.

6.3 Simulator description and implementation

All the distribution strategies described previously have been implemented in a C++ simulator. As already mentioned, the simulator builds an overlay network topology where a single peer (called the *source*) begins sending its information data, divided into k packets, to all the other peers. At the beginning, the source state is set to SEEDER while all the other peers are set to WAIT. The simulator operates on a time slot basis. During each time slot all the network nodes run the selected distribution protocol. Each node is characterized by a maximum upload B_u and download B_d capacity, which determines the maximum number of coded packets that a node can upload/download in a single time slot. The simulator does not assume any form of rate control, thus each node behaves independently, aiming at saturating all its upload capacity. The packets are uploaded with a round-robin scheduling without any feedback from the recipients. On the receiver side, each node can download packets up to the saturation of its download bandwidth; the packets received in excess of the B_d limit are simply dropped.

The simulator is based on a complete implementation of the LT encoding and decoding procedures. As a source, each node has its own random generator for the RSD distribution and can generate the required number of coded symbols in each time slot, based on linear combinations of the original k information packets. As a receiver, each node progressively decodes the received packets. Both the standard BP decoder and the OFG algorithms are available. In the OFG case, each node progressively constructs the generator matrix G, based on the generating equations of the received coded packets. The OFG decoder is run in every time slot to retrieve the maximum number of source packets x_i, given the currently received coded symbols. As soon as a node successfully decodes the original k information packets it signals such event to all its seeding nodes so as to stop them from pushing more coded packets.

The simulator goal is the measurement of the following performance indexes for each node p that is reachable from the source in d hops:

- $t_F(p, d)$: time slot of the first packet arrival, i.e. transition from WAIT to DECODER;

- $t_D(p,d)$: time slot when LT decoding is fully accomplished (the k information packets are recovered), i.e. when a node state switches from DECODER to SEEDER.

- $t_S(p,d)$: time slot when a SEEDER turns OFF, i.e, all of its neighbors completed decoding;

- $\bar{\epsilon}(p,d)$: LT code overhead estimated at node p, i.e. percentage of packets in excess of k downloaded while in the DECODER state. In other words, the number of downloaded packets is expressed as $(1+\bar{\epsilon})k$. Originally, the term overhead has been introduced in the rateless coding literature to identify the penalty due to random linear coding; with abuse of notation we use it to sum up the sub-optimality of both the LT encoding and the distribution strategy.

Previous indexes are then averaged on all peers at given distance d from the source, allowing us to analyze the behavior of the different distribution strategies. From $t_D(p,d)$ we obtain $t_D(d) = \frac{1}{|\mathcal{T}_d|}\sum_{p\in\mathcal{T}_d} t_D(p,d)$ where $\mathcal{T}_d$ is the subset of nodes in $\mathcal{T}$ d hops away from the source. As a performance index for all the overlay we can compute $t_D = \frac{1}{N}\sum_d \sum_{p\in\mathcal{T}_d} t_D(p,d)$ where N is the number of nodes in the graph $\mathcal{T}$. $t_D(d)$ and t_D are termed *absolute decoding time* and represent the average delays between the instant of time when the source has initiated the data distribution and the retrieval of the information. In the case of a P2P live video streaming application, such an index determines the play-out delay. Analogous averages can be computed on $t_F(p,d)$ and $t_S(p,d)$. In particular, as pointed out in the next section, it is interesting to analyze the behavior of $t_D(d) - t_F(d)$, termed the *relative decoding delay*, which represents the average time between the reception of the first coded packet and the complete LT decoding, i.e., the time spent in the DECODER state at d hops from the source.
All protocols have been evaluated on static topologies $\mathcal{T}$, based on the representation of the set of N active nodes in a P2P network as a finite graph of size N, where a vertex represents a peer and application-level connections between peers are modeled as edges.

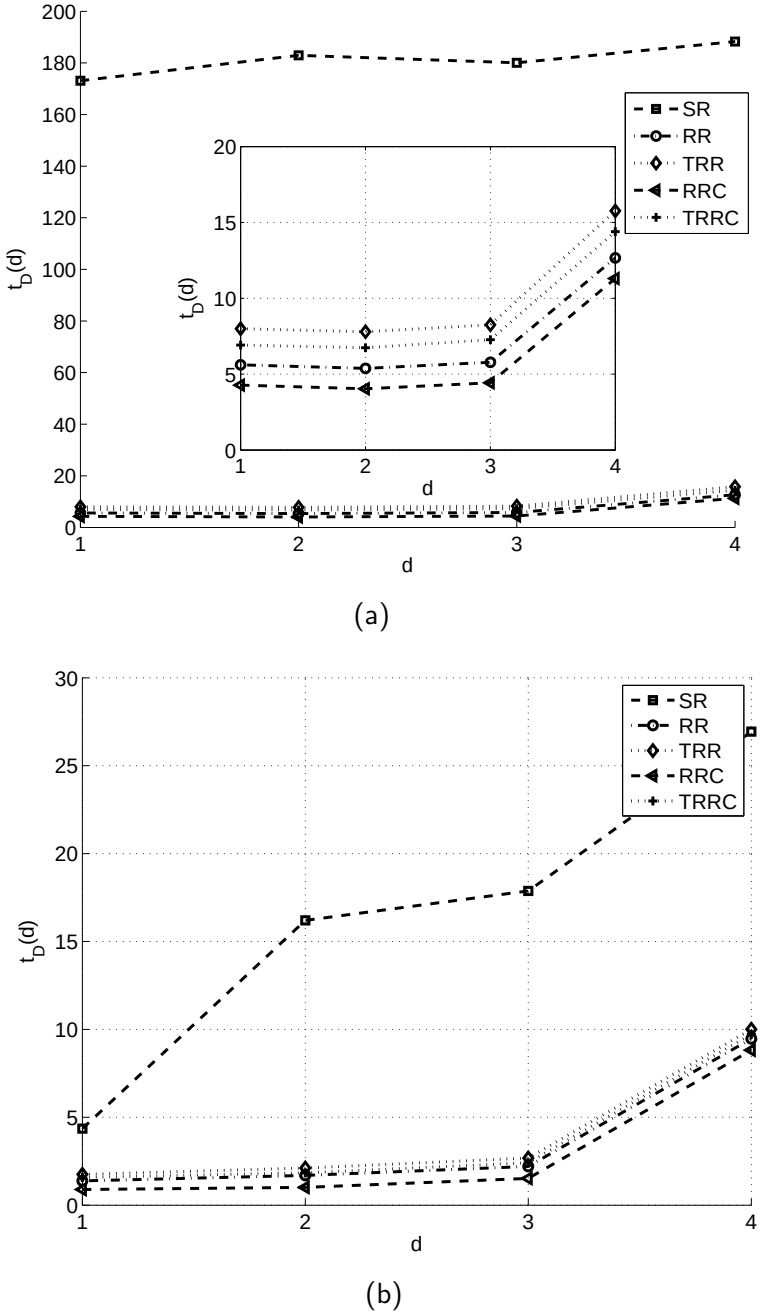


Figure 6.1: $t_D(d)$ for PPLive when source has $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b).

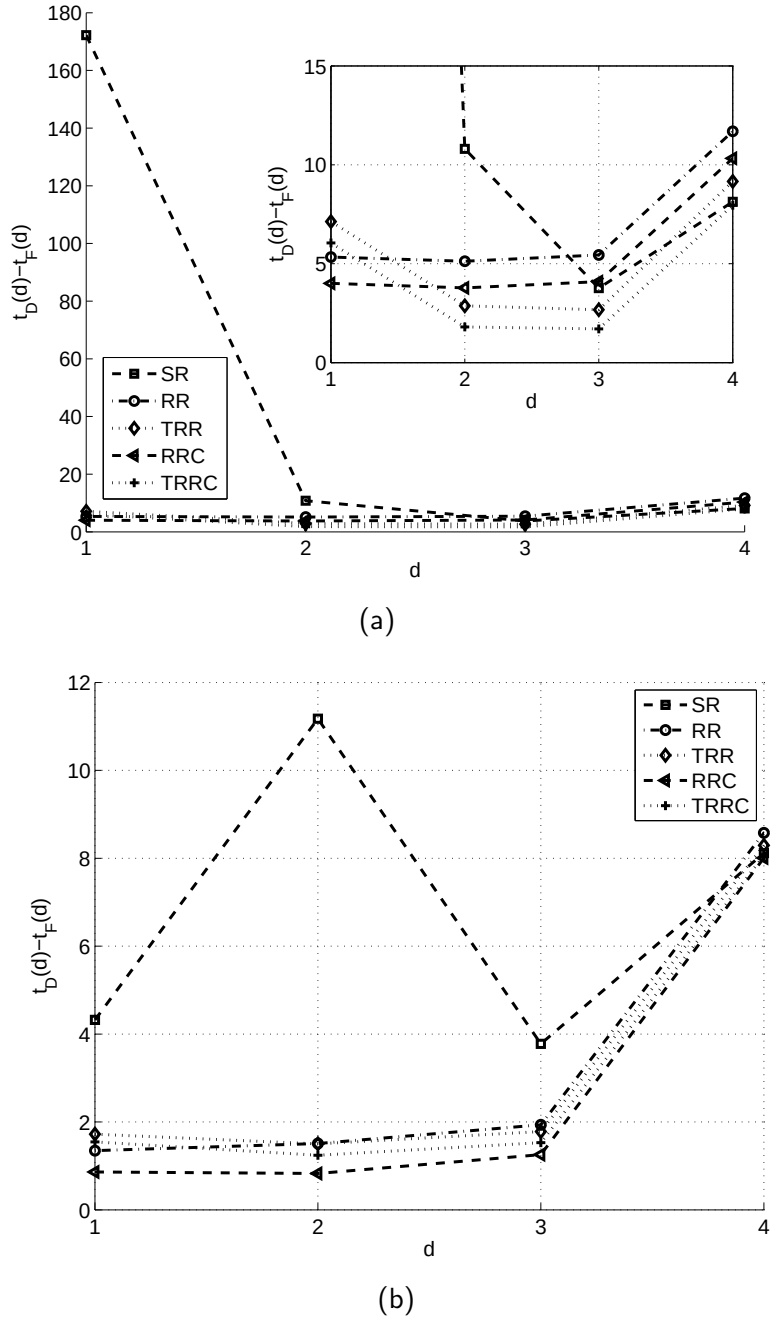


Figure 6.2: $t_D(d) - t_F(d)$ for PPLive, $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b) source.

6.4 Simulation results

In this section the SR (41), RE (4), RR, TRR, RRC and TRRC strategies are analyzed and compared. The goal of this analysis is twofold. On one hand, the advantages offered by relaying and combining coded packets are shown. On the other hand, the experiments contribute to get a deeper understanding of the behavior of protocols based on rateless codes in a complex random network. The simulator described in Sect. 6.3 has been used to simulate the temporal behavior of the system with a time slot equal to 30 ms. Information packets of 1000 bytes and LT coding with $k = 100$, $c = 0.05$ and $\delta = 0.01$ are used. When testing the TRR and TRRC strategies the parameter $\alpha = 2.0$ is used, unless otherwise stated.

For the generation of the network topology $\mathcal{T}$ we considered several instances of Erdős-Rényi (ER) random graphs (42), which are described by a Poisson probability distribution for both the outgoing and incoming degree whose average is equal to z. Another set of experiments has been worked out on real topologies obtained from PPLive video streaming application (44). PPLive peers organize in an overlay to receive and relay multimedia content for a particular channel. We used the crawler (43) to gather topological information of PPLive channels. Because of the overlay dynamics, the accuracy of the captured snapshots depends on the crawling speed. The crawler in (43) reduces the crawling time by using a distributed approach that allows one to capture snapshots of the overlay supporting a PPLive Cartoon channel in times ranging from 5 to 8 minutes. The size of captured snapshots varies according to a daily behavior ranging from 4000 to 8000 peers. We selected 25 snapshots with an average size of 6300 nodes with an average number of neighbors per peer equal to 46. In order to compare the results with those obtained on random graph we used 25 ER graph instances with the same average size and $z = 46$. All the performance indexes reported in the following are obtained by averaging the results of independent simulations on the available 25 graphs. In each simulation the source is randomly chosen among the nodes from which it is possible to reach all the other $N - 1$ nodes. The bandwidth distribution shown in Tab. 6.1 has been used to randomly allocate the peer bandwidth capacities B_u, B_d. The selected bandwidth figures are representative of a realistic scenario with a majority of peers with limited capacities, e.g. accessing through ADSL links, and a small percentage of high capacity institutional nodes. As already mentioned, in this work we

Table 6.1: Bandwidth classes.

B_u	B_d	Percentage
10 Mbps	10 Mbps	5%
256 kbps	2 Mbps	15%
256 kbps	4 Mbps	60%
512 kbps	8 Mbps	15%
1 Mbps	20 Mbps	5%

Table 6.2: Comparison between OFG and BP using SR on ER graphs: source with $B_u = 10$ Mbps.

	t_F	t_D	t_S	$\bar{\epsilon}$
BP	17.75	31.23	32.86	0.35
OFG	14.90	25.71	26.97	0.07

propose to use OFG to decode LT codes. This algorithm has a number of important features, the most important being the possibility of incrementally solving the system of linear equations determined by LT coding, its lower overhead for short block lengths k and its limited sensitivity to the degree distribution of the coded packets. In Tab. 6.2 the average performance indexes t_F, t_D, t_S and $\bar{\epsilon}$, obtained when using the SR strategy and a source node with $B_u = 10$ Mbps on ER graphs, are shown for BP and OFG. As opposed to OFG, BP is very sensitive to the RSD parameters; in the BP case we noticed that using $c = 0.05, \delta = 1.0$ yields the best performance for $k = 100$, shown in Tab.6.2. Note that while the BP decoder requires a significant overhead $\bar{\epsilon} = 0.35$, which means that about 135 coded packets are needed to retrieve the original 100 packets, OFG needs only 107 coded packets. Clearly, the lower overhead positively impacts on all the other indexes, reducing by about 18% the average decoding delay. Moreover, the BP algorithm is not usable in a scenario where LT encoded packets are combined: indeed, the degree distribution of received packets is different from RSD, making BP algorithm unusable. Therefore, all the experimental results reported in the following are worked out with OFG. After this preliminary comparison, we pass to the most important result of this work, i.e. the performance evaluation of the proposed random relay strategies. In Tab. 6.3 the average performance indexes obtained with SR, RE,

Table 6.3: Average performance indexes as a function of the upload capacity B_u of the source.

		$B_u = 256$ kbps				$B_u = 1$ Mbps				$B_u = 10$ Mbps			
		t_F	t_D	t_S	$\bar{\epsilon}$	t_F	t_D	t_S	$\bar{\epsilon}$	t_F	t_D	t_S	$\bar{\epsilon}$
ER graphs	SR (4)	170.95	183.20	184.48	0.07	50.99	62.21	63.48	0.07	14.90	25.71	26.97	0.07
	RE (41)	78.33	123.06	126.86	0.08	20.76	35.48	37.66	0.08	2.89	7.57	8.27	0.08
	RR	0.20	5.41	5.54	4.45	0.18	2.92	3.12	1.93	0.15	1.77	2.04	0.82
	TRR	2.86	5.94	6.12	1.55	1.23	3.04	3.29	0.50	0.67	2.13	2.41	0.27
	RRC	0.20	**3.77**	3.84	2.74	0.18	**1.83**	2.06	0.82	0.15	**0.99**	1.31	0.08
	TRRC	2.85	4.11	4.40	**0.09**	1.23	2.41	2.71	**0.07**	0.66	1.84	2.14	**0.07**
PPLive	SR (4)	173.92	181.39	182.72	0.07	49.08	55.58	56.90	0.07	11.47	17.65	18.97	0.07
	RE (41)	15.79	77.12	77.38	0.08	4.76	22.94	23.19	0.08	1.10	5.61	5.86	0.08
	RR	0.36	6.09	6.49	3.74	0.32	3.49	3.98	1.51	0.29	2.50	3.06	0.68
	TRR	5.43	8.53	9.08	0.61	1.47	3.92	4.48	0.38	0.86	2.95	3.56	0.20
	RRC	0.36	**4.75**	5.10	2.52	0.32	**2.86**	3.35	0.93	0.28	**1.81**	2.41	0.12
	TRRC	5.38	7.49	8.08	**0.08**	1.43	3.29	3.90	**0.09**	0.84	2.68	3.29	**0.09**

RR, TRR, RRC, TRRC are shown as a function of the source upload capacity for both the ER graphs and the PPLive snapshots. It can be noticed that all the proposed RR, TRR, RRC and TRRC strategies, being able to increase the exploitation of the upload bandwidth, reduce tremendously all the delays. On the other hand, random coded packet forwarding can be heavily penalized in terms of overhead, due to the high probability to create duplicated packets in the overlay. In particular when the source upload is limited, RR turns out to be practically unusable with $\bar{\epsilon} \gg 1$. As expected, using packet combinations in RRC significantly improves both in terms of decoding delay t_D and $\bar{\epsilon}$. This is clearly due to the lower probability of forwarding duplicated packets. It can be observed that in all the reported experiments, RRC is the algorithm achieving the lowest t_D, shown in bold face in Tab. 6.3. RRC turns out to be efficient in terms of $\bar{\epsilon}$ only when the source has $B_u = 10$ Mbps. The performance impairment in the case of a source with limited upload capacity can be overcome by throttling the upload resorting to TRRC, which consistently yields the best $\bar{\epsilon}$ (boldface in Table). This advantage is paid in terms of slightly larger delays. TRR, i.e. the same upload reduction but without combinations, still yields large delays and overhead even if it performs better than RR.

To get a better insight into the behavior of the various strategies, we analyze the av-

erage $t_D(d)$ and $t_F(d)$, i.e. the delays as a function of the number of hops separating a node from the source. From this point on all the reported results refer to the case of PPLive snapshots. ER graphs show very similar behavior and they are omitted for conciseness. In Fig. 6.1 $t_D(d)$ is shown as a function of d when the source upload bandwidth is $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b). As already observed in terms of average t_D, RRC is the strategy yielding the lowest decoding delay at any distance from the source and independently of the source upload.

In Fig. 6.2 the relative decoding time $t_D(d) - t_F(d)$ is shown as a function of d when the source upload bandwidth is $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b). In Fig. 6.2(a) it can be noted that SR is heavily conditioned by the limited upload of the source when $d = 1$. Indeed, in the SR case, nodes at distance $d = 1$ are directly served by the unique source; the other strategies, being based on relay of coded packets, let the information flow as soon as possible, thus reducing the LT decoding time. This advantage becomes less evident for higher distances since in such a case both SR and the other techniques rely on the presence of a number of nodes in the SEEDER state, that act as a set of independent sources. The number of such source nodes clearly increases with the distance d. In the reported results, such source propagation effects diminish for $d = 4$ because of the particular topological structure of the PPLive overlay. Indeed PPLive snapshots are formed by very well connected nodes (average degree is 46) when $d \leq 3$. Nodes with $d > 3$ form low connected chains where packet upload can be limited by a single bottleneck. In Fig. 6.2(b) we show the same indexes when the source has a large upload; in such a case the SR relative delay for $d = 1$ is less penalized by the single source, whereas the values for $d > 1$ are the same as in 6.2(a). In the case of RR, TRR, RRC and TRRC the larger upload capacity of the source yields a relevant reduction of the relative decoding delays for all values of d (except for $d = 4$ due to the mentioned topological issues).

Furthermore, TRRC achieves a high utilization of the upload capacities defined as $\eta_u(p, d) = \frac{R_u(p,d)}{B_u[t_S(p,d) - t_F(p,d)]}$, where $R_u(p, d)$ is the total amount of data uploaded by peer p placed at d hops from the source, averaged over all peers belonging to the same bandwidth class. In all the simulated scenarios TRRC exhibits an average η_u above 0.8 for ADSL links, and η_u of about 0.5 for the institutional nodes. As a reference, SR achieves η_u around 0.2 for all bandwidth classes.

6.5 Conclusions and future works

The performance of data distribution in P2P networks can be improved tremendously at a very reasonable cost using rateless codes and RNC. In particular, we propose to perform the linear combinations of information packets in GF(2), to reduce the computational complexity, and to use LT codes to pre-encode the information packets. We let nodes propagate encoded packets as soon as possible, thus increasing the utilization of the upload capacity and reducing the delay after which a node is able to accomplish the decoding, thus retrieving the original information. We show that letting a node relay linear combinations, even if in GF(2), of the coded packets accumulated during the decoding process is very likely to reduce the amount of useless information, thus improving the overall system performance in terms of both delay and bandwidth utilization. The simple random relay approach is potentially dangerous because it is very likely to saturate the upload bandwidth with duplicated packets. To solve this problem, we propose to initially throttle the upload bandwidth. The results are promising: the improvement of the system performance is shown by means of a very detailed simulation of an overlay network of nodes running encoding and decoding stages. The overlay networks we considered are both random graphs and snapshots of PPLive. The proposed distribution strategy yielded very low decoding delay, thus sustaining a larger throughput, while not flooding the overlay network with useless coded packets.
The RNC evaluation of Chap. 4, however, showed that the complexity of the proposed system should be carefully controlled: in fact, the proposed system has a lower computational complexity compared to RNC, but for larger values of k this advantage could vanish. In order to cope with the complexity issue, we propose in the next chapter a novel streaming protocol that uses Band Codes to control the computational complexity.

Chapter 7

Band Codes for Controlled-Complexity RNC

7.1 Introduction

Network coding (NC) was originally proposed for the purpose of improving the network throughput in multiple-source, multiple-receiver scenarios. The distinctive feature of network coding is that the intermediate nodes output linear combinations of received packets rather than just forwarding them. Once the receiver has collected enough linearly independent packets, the receiver solves a system of linear equations to recover the original message (54). The seminal work of Ahlswede inspired and fostered further research that explored both the theoretical and the applicative aspects of network coding. Chou was the first to propose a scheme (55) that made network coding feasible also on realistic packet networks while avoiding the need to know the network topology and the coding functions at the nodes. The authors of (6) addressed the case of wireless communications by proposing a scheme where the network nodes mix packets originating from different sources. This demonstrated that network coding actually improves the throughput of the network. Network coding also found application to the case of peer-to-peer content distribution, because it enabled the design of simple yet effective protocols that overcame issues typical of peer-to-peer systems. Designs such as R^2 (3) showed that the benefits of the network coding for peer-to-peer video architectures are higher quality and lower startup video delays.

The advantages of network coding come, however, at the price of additional compu-

tational complexity, which develops into an issue when power-constrained applications are considered. Battery operated equipment such as mobiles and sensor network nodes just lack the computational resources required to perform full-fledged network coding, while the amount of energy stored in the site battery limits the operational lifetime. An experimental investigation with high-end mobile phones (56) demonstrated that the computational requirements of network coding resulted in throughput bottlenecks compared to the bandwidth achievable by wireless protocols. The study also confirmed that network coding dramatically increased the average energy consumption, hence shortening the battery life.

Previous research in the field of error correction codes resulted in the design of rateless codes that are suitable for reducing the complexity of network coding. Rateless codes such as the Luby Transform (LT) (9) or the Raptor codes (13), for example, are known to achieve good performance together with low decoding complexity. We recall that the key to attaining low decoding complexity is that all the coding operations are performed on a binary Galois field GF(2), which avoids having to perform computationally expensive multiplications. Furthermore, the degree distribution Ω_d, i.e. the probability that the encoder creates a packet combining d blocks of data, is carefully selected so to control the decoding complexity.

When rateless codes are employed in a network coding framework, however, the recombination of the packets at the nodes alters the packet degree distribution and therefore low decoding complexity cannot be guaranteed any longer. To overcome this issue, the P2P streaming application in (4) uses LT codes but avoids recombinations at intermediate nodes. In (38) it is proposed to modify the degree distribution so that the RSD can be enforced at the output of a relaying node combining packets from several sources. Unfortunately, such an approach cannot be easily extended to general relay topologies.

Because nodes with asymmetric computational capabilities should be able to coexist within the same network, the even more challenging question arises of how to modulate the complexity of network coding and to share it among the nodes. It is desirable that the complexity of network coding be controlled independently at each node, so that the complexity can be diverted to nodes with higher computational capabilities. To this end, one must to realize that the computational complexity of network coding stems both from the decoding and the recoding of the packets at the nodes. A possible

approach to the problem is to place the burden of packet recoding on a few ad-hoc deployed nodes, while low complexity nodes simply forward packets without recoding (57). Such an approach addresses only the issue of controlling the packet recoding complexity, however, and enables one to control the complexity with only coarse granularity due to the simple recode-or-forward approach. The problem of jointly controlling both the complexity of the decoding and the recoding is addressed in (58). In particular, the decoding complexity was centrally controlled at the encoder by selecting an appropriate packet encoding degree. An issue with packet recoding at the nodes, however, is that it modifies the selected degree distribution. Hence the average decoding complexity tends to increase hop by hop to an asymptotic value, as shown in Chap. 4. We show in this chapter how the complexity of network coding can be independently controlled at each network node thus avoiding the issues related to packet recoding using Band Codes. This preserves the packet degree distribution through recoding at the nodes. Band Codes boast the further property that their decoding complexity grows linearly with the size of the encoding window, hence with the average degree of the decoded packets. This property enables us to accurately model their decoding complexity so that it can be controlled at each node given a budget for the computational resources. The computational complexity due to packet recoding is further reduced by exploiting the decoder as a prerecoding stage that feeds linearly independent packets to the recoder for better integration of the transmitted packets. We experiment with Band Codes in a peer-to-peer video streaming framework based on network coding. We show that its computational complexity can be controlled to permit operating the network nodes over a wide range of computational complexity levels. Furthermore, we experiment with a scenario where resource constrained nodes (e.g.: battery operated devices) are also considered. We show that controlling the computational complexity of the nodes makes it possible to extend their lifetime.

7.2 Network Coding with Rateless Codes

In this section we present an overview of the functioning of a network coding architecture designed around rateless codes operating on GF(2). We demonstrate that packet recombination at the intermediate nodes results in packets with high decoding complexity that nullifies the advantages offered by rateless codes.

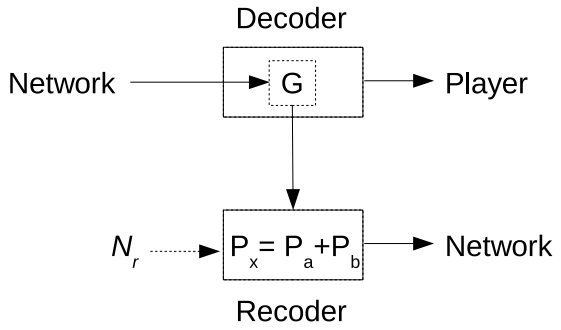
We consider a network scenario where one source node holds the original content and multiple receiver nodes need to receive it. The source node, also known as the *encoder*, distributes the content in chunks of data called *generations*. Each generation is a sequence $(M_0, ..., M_{N-1})$ of N data blocks of identical size B_s, where N is the *generation size*. For each packet to be transmitted, the encoder selects a subset of d blocks, where d is known as the *packet degree*. The set of indices of the selected blocks is known as the *encoding vector* and, in the case of $GF(2)$, it is represented as the bit array $g = (g_0, ..., g_{N-1})$, where the number of non-zero elements is d. The encoding vector and its degree are chosen according to the distribution of Band Codes. The encoder linearly combines the selected blocks and produces the encoded data block $X = \sum_{i=0}^{N-1} M_i \oplus g_i$, where the sum operator is the bitwise XOR. Every transmitted packet $P(g, X)$ thus contains the encoded block X as a payload plus the corresponding encoding vector g as header (55).

The other nodes of the network encompass the functionalities of a *decoder* for the purpose of reconstructing the original content and the funtionalities of a *recoder* to share the content among them. The decoder processes the received packets by inserting them in a decoding matrix G using the OFG decoding algorithm (15). Once N linearly independent packets are received, it is able to solve the corresponding system of linear equations and rebuild the generation. The received packets are stored inside the input buffer and fetched by the decoder.

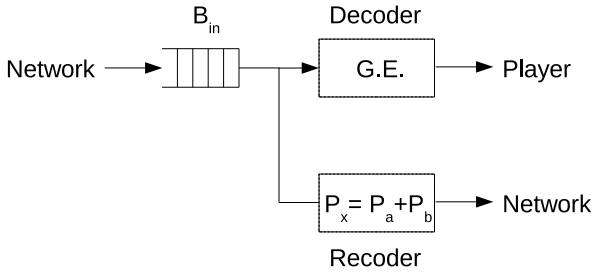
The recoder periodically recombines the packets stored in the input buffer, producing new packets that are transmitted to the other nodes. Packet recoding serves the purpose of increasing the likelihood that the transmitted packets are linearly independent from those already received by the recipient and thus help it to decode the generation. In the case of GF(2), recoding two packets amounts to separately XOR the payloads X and the encoding vectors g. The recoding algorithm will be examined in the following sections.

7.3 Network Coding with Band Codes

In this section we present a streaming protocol that uses Band Codes (BC) to implement a RNC-like system through packet recombination using a novel recombination strategy. Peer nodes are designed around the architecture shown in Figure 7.1(a), which enables



(a) Complexity-Adaptive Node Architecture.



(b) Traditional RNC Node Architecture.

Figure 7.1: Node Architetcure: the difference between the proposed solution and RNC)

.

each node to independently adjust its recoding complexity C_R by selecting the number N_r of packets recombined at each transmission, according to its own computational capabilities. As the linear independence of the recoded packets depends on N_r, however, such a parameter cannot be reduced without impacting on the encoding overhead. The problem of reducing N_r without compromising the linear independence of the recoded packets hence arises. Thus the recoder in the figure recombines rows of the G matrix for the purpose of exploiting the Gaussian Elimination process as a pre-recoding stage. Using OFG algorithm, rows of G represent independent linear equations for the corresponding generation, thus recombining rows of G produces packets that are still linear combinations of the original data blocks. Rows of G undergo a recombination, however, every time a collision happens during the execution of OFG. Because rows of G are recombined at each collision during packet decoding, it is possible to reduce N_r at the encoder without causing bloating of the encoding overhead. Recombining packets from G also enables one to simplify the RNC traditional node architecture,

shown in Figure 7.1(b), dropping the input buffer, as it is not necessary to store the received packets.

The *recoder* recodes the received packets and produces new packets that are transmitted to the other nodes. At each transmission, the recoder recombines N_r received packets selected from the input buffer B_{in}, recombines the packets through repeated XOR operations and transmits the recoded packet. We propose a design where the recoder has an option to recombine rows of the G matrix rather than the received packets (*packet precoding*). Each row of G is in fact either a received packet or the result of an XOR between received packets. Combining rows of G produces packets that are still linear combinations of the original data blocks. They help the receiver to decode the generation. Because OFG already performs several XORs between the rows of G, however, recombining rows of G increases the likelihood that a recoded packet is innovative (or, conversely, smaller N_r is required to produce innovative packets). Finally, because we already demonstrated in prop. 3 that rows of G are band codes, these rows could be used in the recoding process. In fact, the packet recoding algorithm implemented at the nodes is formalized as Algorithm 3. It operates as follows. Given N_r and W as external inputs, one of the $N - W + 1$ possible encoding windows is randomly selected according to the HD distribution. Ψ represents the set of rows of G that are $BC(N, W)$ and whose first index is found at position f. The elements of Ψ are then randomly sorted and the first N_r elements are recombined together into packet P that is transmitted over the network. Following this procedure, the combination of received packets and rows of G produces new BC packets. This permits controlling the average degree of the packets in the system; indeed, from proposition 3 it follows that:

Corollary 1. Using Band Codes and the OFG decoding algorithm in a RNC system where the nodes perform the recoding technique proposed in Alg. 3, the average limit degree distribution of the system is $\mathbb{E}(\Omega^{\infty}) = \frac{W}{2}$.

Proof. During the re-encoding process, packets are xored among them if they share the same window. In practice, the operations are made in windows of length W. However, eq. 4.1 still holds: the degree distribution of the packets will collapse to Dense Codes. The difference is that the collapse is concentrated in the window. Only the degree distribution in the window converges to DC, and in particular to DC of length W; this is exactly the definition of the encoding vectors of Band Codes, hence the re-encoded

packet is indirectly generated following the degree distribution of BC. The expected degree of a BC of window size W is $\frac{W}{2}$. □

Algorithm 3 The Packet Recoding Algorithm

```
select N_r, W
cnt ← 0
P ← NULL
select f with the Horn Distribution
Ψ ← rows of G that are BC(N, W) and first index = f
randomly sort Ψ
while cnt < N_r do
    P ← P ⊕ Ψ[cnt]
    cnt++
end while
transmit P
```

7.3.1 Complexity Model

We analyze the performance of a network coding architecture designed around band codes and we show that there is an inherent tradeoff between encoding overhead and computational complexity. We already showed that the complexity of decoding Band Codes with the OFG algorithm grows linearly with the window size W, which enables us to propose a precise model of the computational complexity of the proposed architecture under the assumption that the recoding algorithm generates new BC packets.

The computational cost C_T of a node is defined as the total number of packets XORed in a second by the decoder (decoding cost C_D) and the recoder (recoding cost C_R) of a node. In the following, we assume that the encoder processes one generation of data per second, so that the cost of decoding or recoding one second of data is equal to the cost of decoding or recoding one generation.

The complexity model in Chap. 4 showed that the cost of decoding band codes with the OFG algorithms linearly grows with the encoding window size. We make the assumption that the recoded vectors belong to BC, and are innovative for the system. We validate this hypothesis by experimenting with different values of generation size N and encoding window size W and comparing the results with the model proposed in

eq. 4.2 for the exact encoding of BC. For the purpose, we encode packets that are fed to the decoder and the actual C_D is measured and compared with the figures predicted by the exact model. Figure 7.2 shows the results of the experiments: a comparison between the curves shows that the exact complexity model can accurately predict the decoding cost of band codes in our re-encoding scenario, supporting our assumption. The *recoding cost* C_R corresponds to the average number of XOR operations between

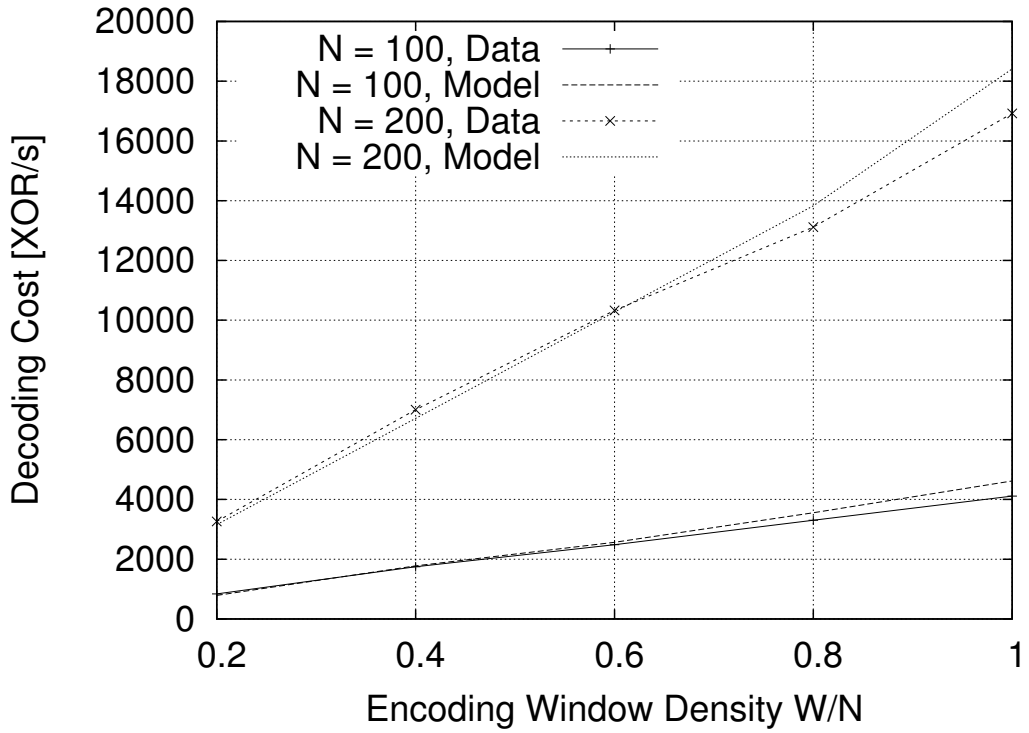


Figure 7.2: Decoding Cost C_D as a Function of the Encoding window Density: Actual Values and as Predicted by the Model.

packets performed per second by the recoder. Assuming that a node transmits N_{tx} packets per second and each packet is the result of the recoding of N_r packets, the recoding cost is

$$C_R = (N_r - 1)N_{tx}. \tag{7.1}$$

The recoding cost is obviously zero if $N_r = 1$, that is if a node forwards the packets without recoding. Thus, the total computational cost of a node is the sum of the

decoding and recoding cost as follows:

$$C_T = C_D + C_R = \frac{WN}{2} + (N_r - 1)N_{tx}. \tag{7.2}$$

7.4 Experimental Results

We experimentally evaluate the performance of band codes in a network coding enabled peer-to-peer video streaming framework. Video streaming is a challenging application due to the constraints on bandwidth and delay that have to be satisfied. To this end, network coding enables the design of simple yet effective *push* protocols that guarantee high throughput and reduced buffering time and thus better perceived video quality. The experiments are performed using the simple peer-to-peer protocol presented in (58). Previous experimental evidence showed that this protocol performs comparable to state-of-the-art competitors despite the simple packet scheduling mechanism it implements thanks to the inherent advantages offered by network coding. While we refer the reader to (58) for a thorough description of the protocol, we overview its main functional aspects and the amendments we introduced to achieve controlled decoding complexity.

A central tracker listens for requests from nodes that want to join the network and replies with a list containing the addresses of the other nodes in the network. The tracker may also assign a specific budget of XOR operations to each node, forcing the node to operate as a complexity-limiting device. The node calculates the corresponding maximum encoding window size W according to Equation 4.2 given the maximum C_D the node intends to bear. Nodes that operate without constraints on the computational complexity specify an encoding window of the same size as the generation. The node starts a handshake procedure with the nodes in the list during which the nodes communicate with each other the selected encoding window size. After the handshake, nodes start exchanging packets using the simple random-push scheduler that serves one packet to one randomly selected node in the network at each transmission opportunity until the bit budget for that generation is consumed. Packets are recombined as shown in Algorithm 3 and transmitted to the selected neighbor. Encoded packets are transmitted using the connectionless User Datagram Protocol (UDP), as network coding is inherently robust to packet losses.

7.4.1 Performance Evaluation of Band Codes in RNC

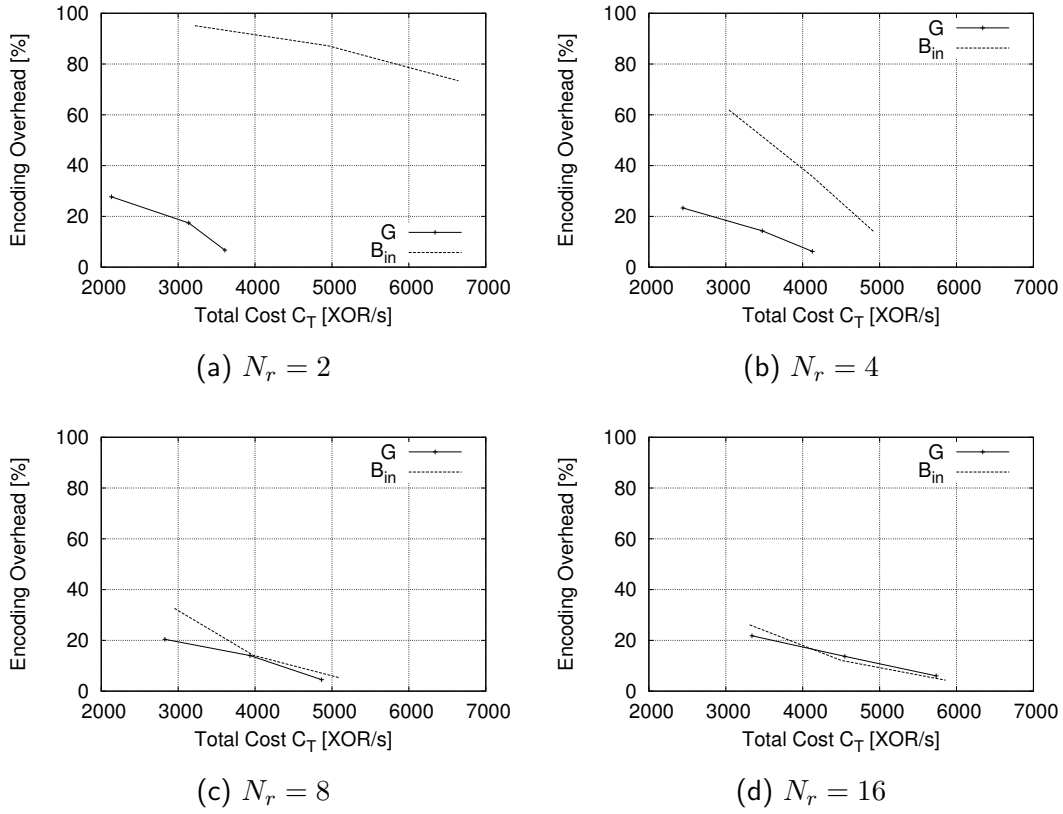


Figure 7.3: Computational Cost C_T and Encoding Overhead ϵ Tradeoffs as a Function of the Encoding Window Size W and the Number of Recombined Packets N_r.

We explore the cost-overhead tradeoffs our proposed architecture can achieve and we show how packet precoding improves the performance of the network coding system. For this purpose, we set up a network composed of 100 nodes that we use to experiment with video streaming. An H.264/AVC video signal encoded at 1 Mbit/s is streamed from the seeder to the peer nodes. Each second of video is encoded as a generation of size $N = 100$, resulting in encoded video packets of 1250 bytes. We experiment with different, setups covering a wide range of cost/overhead tradeoffs. We consider different recoding window sizes $w \in [N/4, N/2, N]$ to control the decoding cost and different numbers of packets recombined at every transmission $N_r \in [2, 4, 8, 16]$ to control the recoding cost. First we experiment with packet precoding, that is we configure the

nodes to recombine packets from the G matrix. Then we compare with a reference strategy where the nodes recombine the packets stored in the input buffer B_{in}. For each experimental setup, we measure the average computational cost of the nodes and the corresponding encoding overhead. The results of the experiments are shown in Figure 7.3 in the form of four graphs, one for each N_r value. Each graph contains two curves, one for the strategy that recombines rows of the G matrix, the other for the reference strategy (B_{in},) while each point on a curve represents a different size of the encoding window. Each graph shows the total cost C_T on the horizontal axis and the corresponding encoding overhead ϵ on the vertical axis.

The figures show that tuning the encoding window size W enables one to effectively control the tradeoff between encoding overhead and the computational cost of band codes. For example, figure 7.3(c) shows that the encoding overhead drops from 20% to below 5% when the window size increases from $N/4$ to N. Conversely, the computational cost decreases from 5000 XOR/s to nearly half such figure when the window size is narrowed from N to $N/4$. Moreover, the figures show that the computational cost increases linearly with the encoding window size as predicted by the model in Eq. 4.2.

A comparison of the two curves contained in each subfigure confirms that packet precoding reduces not only the encoding overhead but also the decoding cost as follows. Precoding achieves low overhead because the rows of the G matrix are linearly independent and hence the higher probability that the recoded packets are innovative. The reduced encoding overhead means that the decoder has to process a lower number of packets to decode, hence resulting in lower C_D.

The figures show that the number of recombined packets N_r should be carefully selected according to the considered packet recoding scheme. If the recoder recombines packets from the input buffer, it is necessary to recombine at least eight packets to achieve an encoding overhead of 5.3%: the more packets that are recombined, the higher the chance that the recoded packet is innovative. On the other hand, recombining two rows of G is enough to achieve a similar encoding overhead (6.7%) at a much lower computational cost. Moreover, increasing N_r to 16 packets does not reduce the encoding overhead in either case, while the total cost grows unnecessarily. Our experiments suggest hencethat the optimum N_r value depends on the considered packet recombination strategy and further research will lead to an analytical formulation.

7.4.2 Resource-Constrained Network Coding

We show that controlling the complexity of network coding extends the lifetime of network nodes in a scenario where power-constrained nodes take part in the communication. We consider a hybrid network composed of two classes of nodes: high complexity (HiC) and low complexity (LoC) nodes. HiC nodes are equipment connected to the power grid such as desktop computers, while LoC nodes represent battery operated terminals like mobiles. In a typical battery-operated system, every packet coding operation draws some energy from the battery until it is exhausted. Clearly, the amount of energy stored in the battery limits the total number of packet coding operations a node can afford and thus the nodes lifetime. We model the level of battery charge as an initial budget of XOR operations that is set at the node startup. Every time the node performs an XOR, the budget counter is decremented by one until it reaches zero and the node leaves the network. We experiment with a network composed of 100 nodes, where a part of the nodes are of the HiC type and the rest are LoC nodes. We stream a 1000 second H.264/AVC video sequence encoded at 1 Mbit/s and we measure the average lifetime of the network nodes, i.e. the average time between a node joining and leaving the network. HiC nodes operate on an unbounded XOR budget, so they recode packets from the input buffer with $N_r = 8$ and with a recoding window size equal to the generation size. Previous experiments showed that such a configuration enables an encoding overhead around 5% for a total cost of about 5000 XOR/s. LoC nodes operate with a limited energy budget that we have set to 2500 XOR/s, that is half the budget the HiC nodes are expected to use.

We evaluate three different complexity adaptation strategies for LoC nodes. The first strategy ($No-Adapt$) is a baseline reference where both LoC and HiC nodes recombine eight packets per transmission ($N_r = 8$) and the encoding window size is equal to the generation size ($W = N$). The second strategy ($Adapt-C_r$) consists in enabling packet precoding for the LoC nodes and reducing N_r from eight to two: such a strategy is expected to reduce the packet recoding cost and corresponds to the approach proposed in (58). The third strategy ($Adapt-C_r-C_d$) improves the previous one as LoC nodes also select an encoding window size equal to N/2 rather than N in order to reduce the decoding cost too. Figure 7.4 shows the average node lifetime as a function of the percentage of LoC nodes present in the network. The reference strategy results in

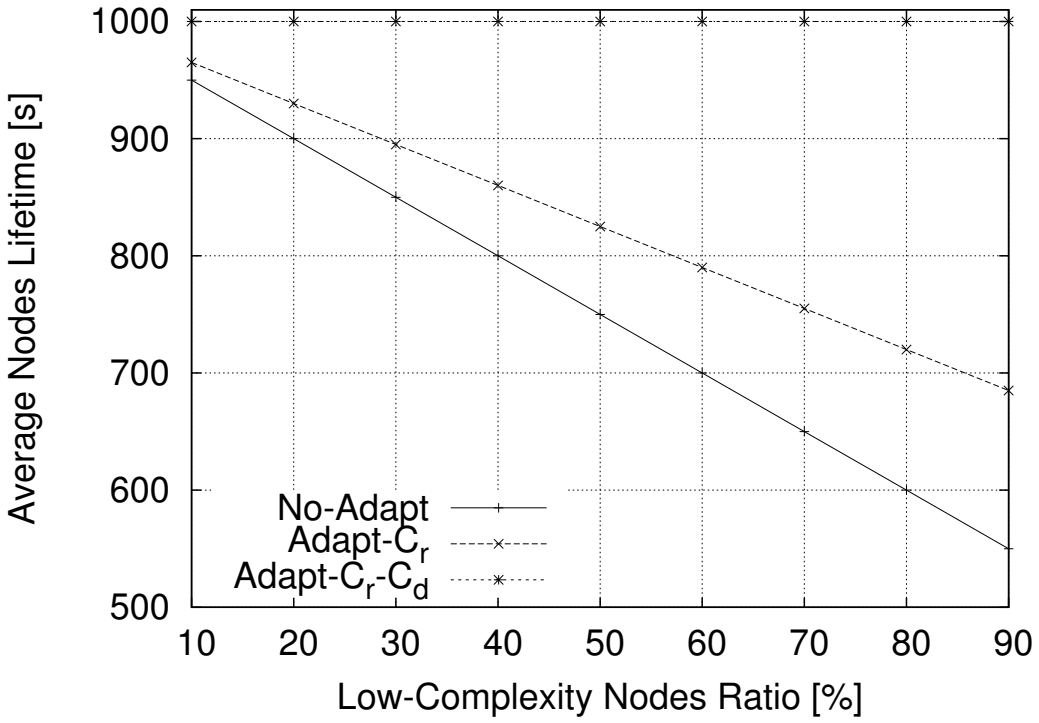


Figure 7.4: Average Lifetime of the Nodes as a Function of the Complexity Adaptation Strategy.

the lowest node lifetimes and the LoC nodes are never able to watch the entire video streaming session. As expected, LoC nodes leave the network roughly at 50% of the streaming time as their XOR budget is half the required budget. The $Adapt-C_r$ strategy increases somewhat the average node lifetime due to the reduced cost of packet recoding. However, the adaptation of the sole recoding cost does not reduce the total computational cost enough to extend the node's life any closer to the total streaming duration. Finally, the joint adaptation of recoding and decoding cost delivers the best results and enables even low complexity nodes to decode almost entirely the video sequence.

7.5 Conclusions

We addressed the problem of controlling the complexity of network coding to meet the asymmetric computational capabilities of different classes of nodes in heterogeneous networks. The computational complexity of network coding is modeled and independently controlled at each node because of a new class of codes, called Band Codes. Band Codes preserve the packet degree distribution selected at the encoder despite the recoding of the packets at the nodes. They thus enable one to control the average decoding cost by appropriately selecting the encoding window size. Furthermore, the computational complexity of packet recombination can be reduced by exploiting the On-the-Fly Gaussian Elimination algorithm to prerecode the packets to be transmitted. We evaluate the performance of the proposed network coding architecture in a peer-to-peer video streaming scenario. This proposed architecture enables a node to operate over a wide range of levels of computational complexity. Hence, it enables nodes with different computational capabilities to partecipate in the network. Finally, we show that controlling the computational complexity of network coding enables one to extend the lifetime of resource-limited devices.

Chapter 8

Broadcast of Multiple Sources of Local Information in Distributed Systems

8.1 Introduction

Data gathering and data aggregation in distributed systems is a challenging problem that has been tackled with several techniques and from different points of view. For instance, in wireless sensor networks sink nodes must have a global view of all or at leat most of the information retrieved and stored by the sensing nodes. In peer-to-peer applications, a global view of the system state may be required by rendez vous points and by ordinary peers to monitor, control and optimize system performance. Research has been succesfull in devising distributed algorithms to provide summaries of global system properties on data that is assumed to be static during certain time interval, so that thw number of properties or the data size is never regarded as an issue, e.g., (45, 46, 48).

In this chapter, we study the more difficult problem of allowing participants to a distributed application to obtain a local view of global system properties. In particular, we assume that each node holds a piece of that information and that any node may require access to the values of the data of all the other nodes periodically at a rate of λ accesses/sec. The goals to be achieved are manifold: first, one wants to guarantee that every node can collect the complete global information in a timely fashion. Moreover,

the communication overhead to accomplish this must be kept as limited as possible to avoid congesting the network. Finally, the processing power of each node must be used parsimoniously.

We propose to use a practical form of network coding based on rateless codes using simple XOR operations that are more computationally efficient than previous attempts based on random network coding in large Galois fields. The proposed approach is based on a continuous flow of control packets being exchanged among the nodes using the *random walk* principle. Each node is allowed to start a limited number of control packets, thus limiting the overall number of random walkers traveling in the network, i.e. thus limiting the communication overhead of the proposed technique. Every node enriches any incoming packet with its local information according to the rateless coding principle and forwards it randomly to one of its neighbors. As a consequence, a continuous flow of packets carrying coded information is spread around the network and any node can use them to recover the global data.

Using network coding in a dynamic scenario where the information varies asynchronously is an issue, since the information combined by each node has to be the same for all random walks to guarantee the inversion of the linear system of equations. Usually this issue is circumvented by constraining the nodes to change the information synchronously, i.e. the information is kept static over a period of time called a generation. In this chapter we present a novel decoding mechanism coping with asynchronous information updates. In particular a novel rateless coding and decoding algorithm is proposed which takes into account the variation of the combined information.

Finally, we show that the efficiency of the proposed solution grows as the size of the data held by each node increases. This occurs in a scenario where communication among nodes is constrained by a limit on the size of the MTU.

The performance of the proposed solution is evaluated both analytically and experimentally by simulation. The analytical analysis proves that the design strategy guarantees a significant reduction in the time required to spread the information among all participants. Moreover, the analytical model proves that the solution scales even better with the increasing size of the network. The simulator shows that the proposed solution is able to cope with a dynamic scenario where the node alternates between active and idle states and the state information varies asynchronously.

The problem of data gathering in a distributed system has been attempted with many

different tools and approaches. The first class of techniques are based on probabilistic gossiping (45, 46). Probabilistic gossiping is generally used to compute a function of the global information, e.g. system averages, and not to actually spread the whole information across a network of N nodes as in this case. Moreover, such techniques rely on a set of assumptions that are difficult to guarantee in practice (47).

Algebraic Gossip, proposed in (48), is the first algorithm addressing data gathering with Network Coding (NC). In this chapter a gossip algorithm based on NC is presented, and we prove that the spreading time of this algorithm is $O(K)$ where $K < N$ is the number of nodes having some information to spread. This algorithm is similar to classical NC: at every transmission opportunity, each node sends to another node a linear combination of the previously received packets computed in Galois Field $GF(q)$ with $q \geq K$. NC exhibits a high computational complexity (40), however, due to the cost of the coding and decoding operations performed in high-order GF. Moreover, each packet requires a padding of additional $Klog_2(q) > Klog_2(K)$ bits. Such padding turns out to be infeasible for large networks. As an example, if $K = 1000$ each packet needs more than 10^4 padding bits. Finally, the authors suggest that the message size m should scale with the size of the network, since it is required that $m >> log(q)$.

A different approach is to store and create packets using rateless codes. In (49) distributed fountain codes are proposed for networked storage. To create a new encoded packet, each storage node asks for information from a randomly selected node of the network. The receiver answers the caller by sending its information. That information will be used by the caller to encode a new packet. A similar algorithm is proposed in (50), where the coded packet formation mechanism is reversed. In this case, the node that stores the information sends random walkers containing the information. The storage nodes store this information and create encoded packets XORing the information they already received. At the end of the process, each storage node stores an encoded packet, and it is possible to retrieve the initial information by querying any $K + \epsilon$ randomly chosen storage nodes.

Growth codes, proposed in (12), use a similar technique but propose a particular degree distribution for the rateless codes to maximize the data persistence in the presence of a single data collector node.

In all the presented papers, the creation of the codes is node-centric, i.e. the nodes cope with the information gathering and the encoding operations. In (51) this responsibility

is assigned to the packets. The aim of this work is to use particular random walks, named as rateless packets, for distributed storage of information in Wireless Sensor Networks (WSN). Each node creates a number of rateless packets that are initially empty packets but that travel across the network as random walks. The goal in (51) is to use packets encoded in a distributed fashion that will be stored at random locations in the network to maximize data persistence in the WSN. Each rateless packet is associated with a degree chosen following the standard LT degree distribution; τ, the mixing time of the graph, is supposed to be known. Each rateless packet performs a random walk across the network and any novel information is combined only one every τ hops. When new information is added, the packet degree is reduced by one. When the degree becomes zero, the random walk performs τ supplementary hops to hit the node that will store the coded packet. The focus of the paper is to increase data persistence. The time required for the distribution of the rateless packets hss not been studied.

Note that in all previous studies, the information to be spread is assumed to be static or alternatively it is divided into generations. Only when a new generation is initiated the nodes are allowed to update their information. The concept of generation represents a significant limitation for practical distributed application since it requires synchronization. The size of the information to be spread, however, is not considered an important parameter of the system.

Finally, another technique adopted for spreading data across a distributed system is based on recent results in the area of compressive sensing (52, 53). These approaches rely on the compressibility of the information and do not apply in the general case when no prior statistical knowledge of the data is available.

8.2 System Description

We model a generic distributed system as graph $G(V, E)$, where V and E are the set of nodes and edges connecting them, respectively. Each node of the network is unambiguously identified by an identifier ID. The ID can be assigned by a fixed rendez vous node, e.g. a tracker, or can be represented by the IP, port address of the node. Each node $v_j \in V$ owns m-bits of information $x_{v_j}^{t_j}$, where t_j is a time-stamp or an integer that is incremented each time the information in v_j changes. To simplify the notation, in the rest of the chapter we assume that v_j coincides with the ID of node; t_j

is usually referred to as the generation number. In our settings, a node can update its information asynchronously with respect to the rest of the network by increasing the generation number associated with the information.

The goal of the nodes is to communicate to one another their respective information so as to realize a concurrent broadcasting of all the information collected by all the nodes in the network. This must be done at an arbitrary rate λ by each node. This observation rules out any centralized solution where all nodes report to a common monitoring node that in turn must propagate the collected data to all the participants. This approach is clearly unfeasible because it imposes a huge amount of traffic towards and from the monitoring node, not to mention the issues related to the election and vulnerability of a centralized sink.

Therefore, we propose a fully distributed solution based on random walks. Each node is allowed to start a limited number w of packets that are the random walkers propagating the information in the network. The parameter w clearly allows one to control the amount of traffic injected in the network. On every reception by a node, the random walk is forwarded to a random neighbor, thus realizing a simple form of probabilistic gossiping. It is well known that network coding solutions, e.g. carrying a linear combinations of the collected information rather than the list of information collected during the random walk, increases the performance in terms of throughput, robustness and persistence (48, 51). On the other hand, coding approaches exhibit two main shortcomings. The first and most studied issue is represented by the added computational complexity. A possible solution that has already been proposed in the literature (51) is to simplify the original random network coding approach, that requires one to combine the data blocks in a high order Galois Field with systems based on simple binary combinations, e.g. XOR. Our work copes with the complexity issue by using LT codes (9). The second most relevant shortcoming of NC is the fact that a node cannot update asynchronously the information it combines without catastrophically impacting on the decoding capability of all the other nodes. Indeed, the nodes keep collecting linear combinations of a set of unknowns until they successfully invert the corresponding system of equations. Clearly, the system of linear equations is meaningful if one keeps combining the same information. On the contrary, we propose a novel decoding approach for LT codes that is resilient to asynchronous changes of the information. We let each node propagate a fixed number of packets carrying coded information of the nodes that

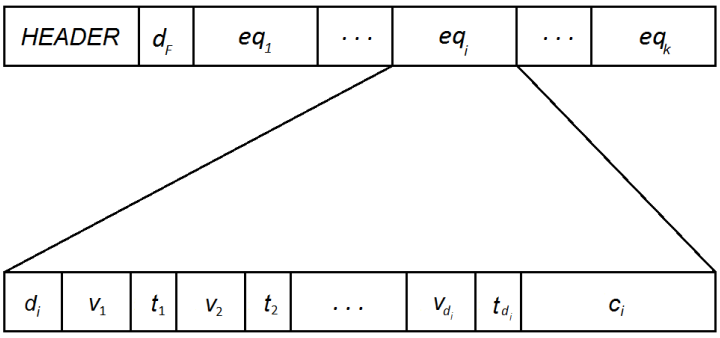


Figure 8.1: Packet format

the packets have hit performing a random walk along $G(V, E)$. All the nodes use the received packets to solve a system of linear equations allowing them to retrieve the data associated with all the information collected by the network in a timely, complete and robust way.

In the following, the details of the proposed random walk coding strategy and the design of the novel LT decoding algorithm are presented.

8.2.1 Random walk LT coding

In this section, we describe the structure of the packet spreading the coded information using a random walk approach. The packet format is shown in Fig. 8.1. Each packet is composed of a header followed by a set of equations $\{eq_1, \ldots, eq_k\}$. Note that one packet carries multiple coded information at a time. An equation eq_i, $i = 1, \ldots, k$, is characterized by the degree d_i, a set of d_i pairs v_j/t_j, $j = 1, \ldots, d_i$, and an encoded m bits message c_i. The degree d_i of the equation corresponds to the number of nodes that have encoded their information $x_{v_j}^{t_j}$ in c_i. The message c_i corresponds to all the information that has been progressively XORed by the nodes hit by the packet. In fact, we have

$$c_i = \sum_{j=1}^{d_i} x_{v_j}^{t_j},$$

where the summation is performed in $GF(2)$.

The coding strategy follows the standard LT approach (9), where the degree of the

equations is constrained to follow the probability density function known as the Robust Soliton Distribution (RSD). That in turn guarantees asymptotically optimal coding performance. In other words, using RSD, it is very likely that each node can recover N coded bits of information, from any set of $N' \geq N$ equations, with an N' that is arbitrarily close to N in the limit $N \to \infty$. This property holds for any information of size m. RSD depends on N, that should be known by the nodes. A weak approximation of N is sufficient, however, to compute RSD without a great loss of performance if a decoder based on Gaussian Elimination is used (25, 34). This approximation could be calculated from the nodes by observing the IDs of the nodes contained in the previously received equations. To cope with the creation of the equation according to the RSD, each packet carries in the header part of the signalling of the degree d_F that must be achieved by the equation under formation in the packet (that in our settings is the first equation written in the packet body from left to right). When a node v_j at generation t_j receives a packet, it checks if the degree of the first equation stored in the packet has reached the requested degree. If $d_F > d_1$, hence the target degree has not yet been reached, the node performs 3 operations: it XORs its information to the term c_1, i.e. $c_1 = c_1 \oplus x_{v_j}^{t_j}$. Then the degree d_1 of the equation is incremented and the corresponding field in the packet is updated. Finally, the node v_j and the information timestamp t_j are appended to the equation. On the other hand, if $d_F = d_1$, the first equation has already achieved the requested degree. Hence a new equation is created and stored as the new first equation, while the other equations are shifted, e.g. eq_i becomes eq_{i+1} for $i = 1 \dots k$. To create a new equation eq_1, a node draws a random degree from RSD and stores it in the d_F field of the packet header. Then $d_1 = 1$ is set, its v_j, its actual timestamp t_j and the information $c_1 = x_{v_j}^{t_j}$ are written in the proper fields. Finally, the equation is stored in the packet as the first equation, shifting all the other equations (if present). Every packet created or updated by a node is then forwarded to a node, randomly selected among the local neighbors.

The number of hops globally taken by a packet is not limited in our system. The only limitation is represented by the maximum packet size DIM, that is generally imposed by the maximum transfer unit allowed by the underlying communication technology at the physical layer. When a packet approaches the maximum dimension DIM, the eldest equation carried by it is deleted since it is very likely to carry old or already known information.

In the case of a dynamic network where nodes can randomly join and leave the graph $G(V, E)$ and in the presence of unreliable links that turn into packet losses a mechanism to acknowledge the presence of a given random walk in the network must be devised. To this end it is possible to insert supplementary control information in the header of the packet. As an example, one can signal in the header the address of the originator and an acknowledgement timer (ACK). The ACK is an integer that represents the number of hops preceding the generation of the next acknowledgement to the originator. The ACK field is initialized to a constant value upon the packet creation, then each node decrements it. The node that is hit by a random walk with ACK= 0 is responsible for signaling to the originator that its random walk is still alive. The ACK is initialized back for the next acknowledgement period. The originator of the random walk uses a timer to detect packet losses. When a timer expires before the reception of the corresponding ACK message, the node is allowed to regenerate the random walk.

8.2.2 Decoding

The information spread by the random walkers can be straightforwardly recovered by any node in the network as soon as a sufficient number of packets has been collected to run a standard LT decoding algorithm (9, 25, 34). Since our goal is to reconstruct the information as quickly as possible, all the equations carried by each packet, including one that is still under formation, are buffered by the nodes. If we assume that the number of nodes $|V|$ in the network is equal to N, the decoder task can be formulated as the solution of the following system of linear equations

$$\mathbf{G}\mathbf{x} = \mathbf{c},$$

where $\mathbf{G}$ is an $N \times N$ binary[1] matrix whose rows represent the N possible independent equations collected by the node, $\mathbf{x}$ and $\mathbf{c}$ are $N \times 1$ column vectors representing the N unknown pieces of information and the corresponding buffered linear combinations carried by the packet payloads. Both $\mathbf{x}$ and $\mathbf{c}$ contains m-bits elements. The node can recover all the information $\mathbf{x}$ using a progressive form of Gaussian Elimination (34) to solve the system of equations. Clearly, this will require all the nodes in the network to

[1]The matrix is binary since LT coding is performed in GF(2) and it is analogous to the code generation matrix of standard forward error correction linear block codes.

keep their information constant to avoid perturbing the solution of the system.
To circumvent this strong limitation, we propose a novel decoding strategy that does not assume that the N pieces of information $\mathbf{x}$ be constant. We assure that the vector $\mathbf{x}$ is extended to the $2N \times 1$ vector $\tilde{\mathbf{x}}$, where the bottom N elements refer to the updated information, e.g. the most recent timestamp for each information block, whereas the top N elements are related to the previous generation. In other words, vector $\tilde{\mathbf{x}}$ represents a snapshot of the information collected in the network with a sliding window mechanism including the two most recent generations of the information for each node. Clearly, this strategy can be generalized to cover more than two generations at the cost of a higher computational cost and memory requirements. To actually implement the sliding window mechanism, two auxiliary vectors are used; an $N \times 1$ vector $\mathbf{a}$, whose i-th element is the ID of the node associated with the information in position i and $i+N$, and an $N \times 1$ vector $\mathbf{t}$ containing the timestamps of the most recent information collected for each node. According to the previous description, it follows that $\tilde{\mathbf{x}}[i] = x_{\mathbf{a}[i]}^{\mathbf{t}[i]-1}$ and analogously $\tilde{\mathbf{x}}[i+N] = x_{\mathbf{a}[i]}^{\mathbf{t}[i]}$, where square brackets are used to index a single element of a vector. Adopting this extended notation, the decoding task can be formulated as

$$\tilde{\mathbf{G}}\tilde{\mathbf{x}} = \tilde{\mathbf{c}}, \tag{8.1}$$

where $\tilde{\mathbf{G}}$ is a $2N \times 2N$ matrix specifying the linear combinations among the two most recent versions of information circulated in the network. This keeps the decoding as updated as possible, aiming at reconstructing the last N elements of $\tilde{\mathbf{x}}$. Nonetheless, the equations referring to the previous version of a given node are not invalidated, but are stored in the upper part of the matrix to enable the exploitation of new incoming equations still involving the older version of the information. In fact, this outdated information can still be present in some random walkers hitting the node.
The objective of the decoder is twofold. First, given all the equations extracted by the random walkers we want to obtain the maximum number of information items. Moreover, this needs to be done in an incremental way. Second, we propose a strategy to manage the extended decoding matrix $\tilde{\mathbf{G}}$ which makes the decoding process robust to asynchronous updates of the information. In general terms the idea is as follows. An incremental version of the Gaussian Elimination algorithm is used to insert the new equations collected in the random walkers, while guaranteeing that $\tilde{\mathbf{G}}$ is an upper triangular matrix and that the maximum number of $x_{v_j}^{t_j}$ are known. This latter condition

corresponds to rows of $\tilde{\mathbf{G}}$ with a single 1 on the diagonal, and can be obtained by the incremental back substitution strategy shown in (34). As soon as a node detects an information update by retrieving an equation using a new timestamp for an information block, the data in the linear system of Equation 8.1 are rearranged to reflect the changes.

The detailed decoding operations are now discussed. When a new packet is received, the equations collected by the packet during the random walk are iteratively processed and inserted in the matrix $\tilde{\mathbf{G}}$. Each equation eq_i is used to create a new row vector g_{new} (of length $2N$) and a new element of $\tilde{\mathbf{c}}$ called $c_{new} = c_i$. Each pair (v_l, t_l), $l = 1, \ldots, d$ in eq_i is checked to determine whether the node v_l is already known or not. If not, the number of known nodes N is incremented. The new node v_l and the timestamp of the information are appended to vectors $\mathbf{a}$ and $\mathbf{t}$, respectively. $\tilde{\mathbf{G}}$, $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{c}}$ are enlarged to make room for the new node. In particular, two new all zero rows are inserted as the N-th and $2N$-th rows of $\tilde{\mathbf{G}}$. Finally we set $\tilde{\mathbf{c}}[N] = \tilde{\mathbf{c}}[2N] = 0$. On the contrary, if v_l is already known and therefore listed in the k-th position of $\mathbf{a}$, the timestamp t_l is compared with the most recent observation stored in vector $\mathbf{t}$. If $\mathbf{t}[k] = t_l$, then the information XORed by the node v_l in c_i is updated for the receiving node; as a consequence we set $g_{new}[k+N] = 1$. If $t_l = \mathbf{t}[k] - 1$ then the information of the node v_l is older than the last observation but still useful; as a consequence it is stored in the left part of $\tilde{\mathbf{G}}$ by setting $g_{new}[k] = 1$. If $t_l < \mathbf{t}[k] - 1$, the information carried by eq_i is considered obsolete. Hence this information is discarded and the process restarts using the following equation. If $t_l = \mathbf{t}[k] + 1$, the information combined in the eq_i by v_l has just changed, and the decoding structures must be updated. Still the information referring to $x_{v_l}^{t_l - 1}$ that was stored in the right part of the matrix should be useful if not already decoded. In this case, the function `Update_decoder`$(k+N)$ (described in the following) is used to move the information from the right to the left part of $\tilde{\mathbf{G}}$ thus making room for the new generation t_l. Moreover, we set $\mathbf{t}[k] = t_l$ and $g_{new}[k+N] = 1$. Finally, if $t_l > \mathbf{t}[k] + 1$, then all the information that was collected previously about v_l is obsolete and can be removed from the system. This is done using `Update_decoder`$(k+N)$ and `Update_decoder`(k). In this case we set $\mathbf{t}[k] = t_l$ and $g_{new}[k+N] = 1$.

Processing all the pairs (v_l, t_l) of eq_i a vector g_{new} with d_i 1s and $(2N - d_i)$ 0s is obtained. This g_{new} along with the corresponding linear combination c_{new} is then inserted in Equation 8.1 using the LT optimal partial decoding algorithm (34). In particular,

g_{new} is used to update the j-th row of $\tilde{\mathbf{G}}$, where j is the position of the leftmost 1 in g_{new}.

The purpose of `Update_decoder`(j) is to remove the obsolete information which the j-th column of $\tilde{\mathbf{G}}$ is referring to if $j \leq N$. Otherwise, if $j > N$ the goal is to update the system so that column j is shifted leftwise taking the place of column $j - N$. `Update_decoder`(j) determines the row g_F (the F-th row of $\tilde{\mathbf{G}}$) that contains the first 1, starting from the bottom, in column j. Since the decoding algorithm keeps $\tilde{\mathbf{G}}$ in upper triangular form, the other 1s of the j-th column can only be found in some of the rows preceding g_F. The function `Update_decoder`(j) cancels all previous 1s of column j by XORing g_F to any row g_l with a 1 in the j-th column. To this end, the operations $g_l = g_l \oplus g_F$ and $\mathbf{c}[l] = \mathbf{c}[l] \oplus \mathbf{c}[F]$ are performed. If $j < N$, row g_F (and the corresponding $\mathbf{c}$[F]) are removed from system (substituting them with all 0s). Otherwise, the information is moved to the left part of $\tilde{\mathbf{G}}$. This requires rewriting row g_{j-N}; if g_{j-N} is not empty, its contents are removed by running `Update_decoder`($j - N$). To create the new row g_{j-N}, the 1 in the j-th column of g_F is moved to position $(j - N)$. Then, we set $\mathbf{c}[j - N] = \mathbf{c}[F]$, and clean up row r_F and the corresponding linear combination $\mathbf{c}[F]$.

8.3 Recovery Time Model

In this section we provide an analytical analysis of the time required to spread all the local information to all the participants in the network. We define that as the *recovery time*. In particular, we are interested in modeling the recovery time as a function of the size of the local information m, the number of random walks generated per node w and number of nodes in the network N, given the constraint on the maximum size of the random walk packets DIM. Moreover, the proposed analytical model permits one to compare the coded approach versus an analogous system without coding, i.e. when the information is gossipped explicitly. In fact, the proposed approach degenerates into an uncoded system if one uses a constant equation of degree equal to one; the packet format of Fig. 8.1 can hence be changed, removing the d_i and d_F fields, that become useless.

Given the size of the transmission packet DIM (in bits), we assume that a packet is divided into two parts: the header and the free space, of sizes h and $f = DIM - h$

respectively. From now on, we refer to the memory size of an object in bits. In every equation, we call g the size of the pair (v_l, t_l) and m the size of the combined message c_i. Following the packet specification in Fig. 8.1, we have that the size of a single equation becomes $e_C = d_i \cdot g + m$, where d_i is the degree of the equation. In the following derivation, we perform a mean value approximation assuming that all equations have the same degree $d_i = d$, where $d = 2 \ln N$ (9) is the average degree of LT codes. In previous approximation, we also considered the cost of signaling the value of d as negligible. Analogously, the size of the message gossipped by an uncoded system is simply $e_U = g + m$, i.e. on every hop a new identity along with the corresponding local information is added to the payload of the packet. If we call n_U and n_C the maximum number of equations storable in an uncoded and encoded packet respectively, we have that

$$n_U = \left\lfloor \frac{f}{e_U} \right\rfloor = \left\lfloor \frac{f}{g+m} \right\rfloor, n_C = \left\lfloor \frac{f}{e_C} \right\rfloor = \left\lfloor \frac{f}{d \cdot g + m} \right\rfloor.$$

If we set $g = 2 \log_2 N$, i.e. twice the number of bits required to map N unique identifiers, both n_U and n_C turns out to be functions of m and N.

We are interested in finding the cumulative number of equations distributed by every packet as a function of the number of hops T, termed as $N_U(T)$ and $N_C(T)$ for the uncoded and the encoded case, respectively. Both functions also depend on the number of equations stored in the packet. In fact, the number of collected equations saturates to a maximum number that depends on the maximum packet size DIM. When this limit is reached, the packet is full and the number of equations remains constant (on average for the encoded case).

For an uncoded packet, $N_U(T) = 1 + 2 + \cdots + T = \frac{T(T+1)}{2}$ if $T < n_U$, i.e. so long as the packet is not full. This happens because a new piece of information is added at the packet at every hop until the packet is full. When the packet is full,

$$\begin{aligned} N_U &= 1 + 2 + \cdots + n_U - 1 + \underbrace{n_U + \cdots + n_U}_{T-(n_U-1)} = \\ &= \frac{n_U(n_U-1)}{2} + (T - (n_U - 1)) \cdot n_U = \\ &= n_U \cdot (T \cdot \tfrac{n_U - 1}{2}). \end{aligned}$$

The previous derivation is less trivial in the case of an encoded packet: in fact, an equation is completed only once every d hops on the average. Hence the packet is full after $T = (n_C - 1) \cdot d$ hops. By setting $h = \lceil T/d \rceil$ and taking into account that in our

proposal partially formed equations are collected by the nodes, when $T < (n_C - 1) \cdot d$ we have:

$$\begin{aligned} N_R &= \underbrace{1 + \cdots + 1}_{d} + \cdots + \underbrace{(h-1) + \cdots + (h-1)}_{d} + \\ &\quad + \underbrace{h + \cdots + h}_{T - d(h-1)} = \\ &= d\frac{h(h-1)}{2} + h(T - d(h-1)) = \\ &= d\frac{\lceil T/d \rceil \lfloor T/d \rfloor}{2} + \lceil T/d \rceil (T - d\lfloor T/d \rfloor) = \\ &= \lceil T/d \rceil \left(T - \frac{T\lfloor T/d \rfloor}{2} \right) \end{aligned} ,$$

The coded packet is full when $T = (n_C - 1) \cdot d$; in this case we have,

$$\begin{aligned} N_R &= \underbrace{1 + \cdots + 1}_{d} + \cdots + \underbrace{(n_C - 1) + \cdots + (n_C - 1)}_{d} + \\ &\quad + \underbrace{n_C + \cdots + n_C}_{T - (n_C - 1) \cdot d} = \\ &= d \cdot \frac{n_C(n_C - 1)}{2} + (T - (n_C - 1) \cdot d) \cdot n_C = \\ &= n_C \cdot (T - d \cdot \frac{n_C - 1}{2}). \end{aligned}$$

We can summarize the results as follows:

$$\begin{aligned} N_U &= \begin{cases} \frac{T(T+1)}{2} & if\ T < n_U \\ n_U \left(T - \frac{n_U - 1}{2} \right) & otherwise \end{cases} \\ N_C &= \begin{cases} \lceil T/d \rceil (T - \frac{d\lfloor T/d \rfloor}{2}) & if\ T < (n_C - 1)d \\ n_C \left(T - d\frac{n_C - 1}{2} \right) & otherwise \end{cases} \end{aligned} \tag{8.2}$$

Taking into account that every node creates w packets, it turns out that the global number of equations spread in a network of N nodes after T hops on average $w \cdot N \cdot N_C(T)$ ($w \cdot N \cdot N_U(T)$ in the uncoded case). If the $G(V, E)$ is regular, i.e. the distribution of the outgoing (and incoming) edges is peaked around the average, the equations hit any node with the same probability. Therefore each node receives on the average $R_C = \frac{w \cdot N \cdot N_C(T)}{N} = w \cdot N_C(T)$ equations ($R_U = w \cdot N_U(T)$ in the uncoded case). Summarizing, we get

$$\begin{aligned} R_U &= \begin{cases} w\frac{T(T+1)}{2} \simeq w\frac{T^2}{2} & if\ T < n_U \\ w n_U \left(T - \frac{n_U - 1}{2} \right) & otherwise \end{cases} \\ R_C &= \begin{cases} w\lceil T/d \rceil (T - \frac{d\lfloor T/d \rfloor}{2}) \simeq w\frac{T^2}{2d} & if\ T < (n_C - 1)d \\ w n_C \left(T - d\frac{n_C - 1}{2} \right) & otherwise \end{cases} \end{aligned} \tag{8.3}$$

where we introducde some linear approximations to simplify the following analysis. Using these approximations, it is possible to reverse the functions obtaining the number

of hops needed to distribute a certain number of equations, i.e. T as a function of R_U and R_C:

$$T_U = \begin{cases} \sqrt{\frac{2R_U}{w}} & if\ R_U < \frac{n_U(n_U-1)}{2} \\ \frac{R_U}{wn_U} + \frac{n_U-1}{2} & otherwise \end{cases}$$
$$T_C = \begin{cases} \sqrt{\frac{2dR_C}{w}} & if\ R_C < d\frac{n_C(n_C-1)}{2} \\ \frac{R_C}{wn_C} + d\frac{n_C-1}{2} & otherwise \end{cases} \tag{8.4}$$

Our objective is to estimate the number of hops (i.e. the time) needed by any node to retrieve all the information present in the network. Equations 8.4 can be used to calculate such a recovery time. For the encoded case, in a network of N nodes it is theoretically sufficient to receive $R_C = N$ equations to enable decoding. For the uncoded case, however, the number of equations turns to be higher; in fact, this can be recognized as an instance of the coupon collector's problem. In the case of equally distributed coupons, every node needs about $R_U = N \cdot (\ln N + \frac{1}{2})$ equations to collect all the information. In this system, however, the probability of receiving a certain equation is not uniform, but rather depends on the length and the number of paths that connect the nodes. In Sect. 8.4 we show that the coupon collector's approximation is a lower bound for the actual performance of an uncoded system. Substituting R_C and R_U in (8.4) one obtains the recovery time as a function of w, DIM, N and n_C (or n_U). We recall that in turn n_C and n_U depend on m and N. In Fig. 8.2(a) the recovery time evaluated according to (8.4) is reported as a function of m for the case $DIM = 1500$ bytes, i.e. a typical value for the maximum size of a UDP packet for a network with $N = 1000$ nodes. The proposed approach is able to spread the information much faster than the uncoded counterpart and that the gap increases for larger information size. In Fig. 8.2(b) the recovery time is shown when m is fixed to 128 bytes and we increase the network size. This results point out that the coded approach scales much better with the network size, making the proposed solution very attractive for large distributed systems.

Finally, the recovery time model can be used to fix the value of the parameter w (the number of packets created per node). Reversing (8.3) it is indeed possible to calculate the minimum value of w that guarantees retrieval of all the information within a time

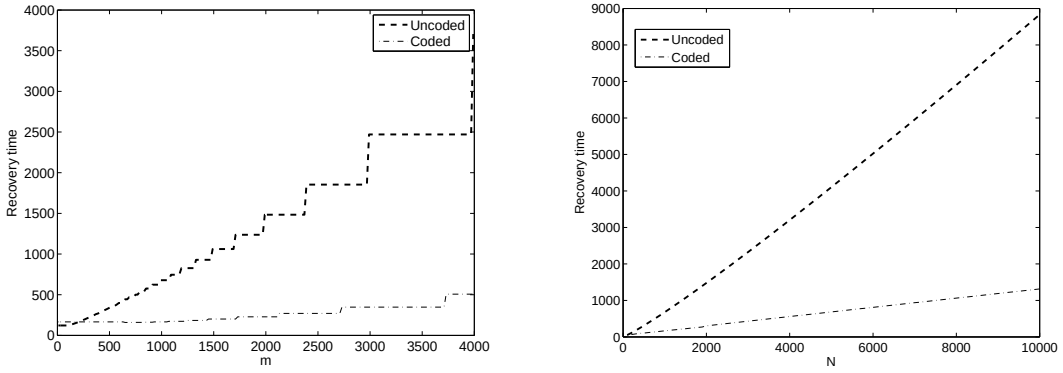


Figure 8.2: Recovery time as a function of m in bits (a) and N (b).

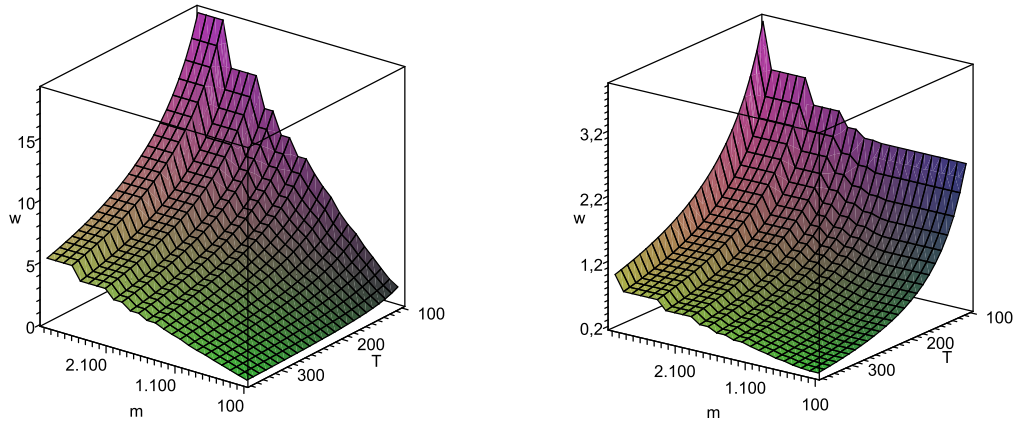
interval T. Such a minimum value turns out to be:

$$W_U = \begin{cases} \frac{2R_U}{T(T+1)} & if\ T < n_U \\ \frac{2R_U}{n_U(2T-n_U+1)} & otherwise \end{cases}$$
$$W_C = \begin{cases} \frac{2R_C}{\lceil T/d \rceil (2T-d\lfloor T/d \rfloor)} & if\ T < (n_C-1)d \\ \frac{2R_C}{n_C(2T-d(n_C-1))} & otherwise \end{cases} \tag{8.5}$$

for the uncoded and coded approaches respectively. Equations 8.5 are shown in Fig. 8.3 as a function of m and T under the hypothesis that $N = 1000$ and $DIM = 1500$ bytes. Again, for large values of m, the advantage of the encoded system is apparent, i.e. the same performance requires much less network traffic in terms of recovery time.

8.4 Simulation results

In this section we present the results obtained through a simulator of the proposed approach and the corresponding uncoded counterpart. The system is simulated by means of an ad-hoc C++ simulator. The simulator creates a random network of N nodes; each node connects to a fixed number of neighbors N_{neigh}, randomly spread among all the participants. The information $x_{v_i}^{t_i=1}$ at each node is initialized to a random value that can be updated asynchronously by incrementing the corresponding timestamp t_i. The simulation time is slotted. At startup, each node initiates w new random walks, sending w packets to a set of randomly chosen neighbors. In each time slot T, the packets received by any node at time $T-1$ are updated with a local piece of



(a) W_U (b) W_C

Figure 8.3: W_U and W_C as a function of m and T.

information and forwarded randomly. The maximum packet size is fixed at 1500 bytes in all simulations. As predicted by the analytical model of Section 8.3, the parameter w has the effect of only to changing the time scale of the results. For conciseness we perform all the simulations for the case of $w = 1$. The network remains static if nodes do not leave or join the network. In the opposite case of a dynamic network, the nodes can join and leave randomly, thus changing the irrespective neighborhood. To simulate a dynamic network, the number of nodes is fixed at N as in the stable case, but the nodes alternate between on and off periods. Nodes in the off state leave the network, causing their neighbors to replace them, thus changing the topology dynamically. Moreover, a node that turns off in time slot T does not forward any packet, received at time $T-1$ eventually. To keep the overall number of random walks in the network constant, ideal singnalling is assumed, so that the originators of the lost packets can restart new random walks at time $T+1$. It is clear that both the replacement of neighbors and the acknowledgement mechanism would be implemented according to a practical distributed protocol, although this adds delay with respect to the ideal behavior of the simulator. Nonetheless, since the goal of our analysis is a relative comparison between the performance of coded and uncoded systems, we are not interested in adding a suboptimality of an actual implementation that would equally affect both approaches.

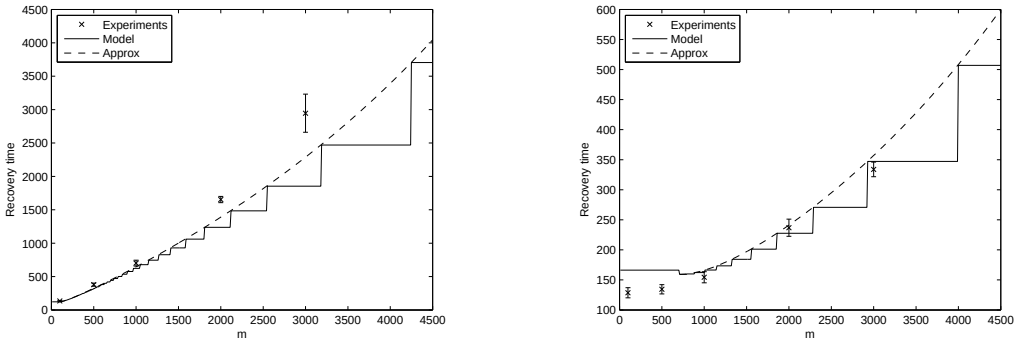
The system performance is measured in terms of the amount of information that can be gathered by the nodes and the time required to achieve the result. In particular, for each node v_l we calculate the percentage of information retrieved by that node as a function of time T:

$$p^l(T) = \frac{\sum_{j=1}^{N} \left(\sum_{i=1}^{t_j(T)} \mathrm{dec}_l(i,j) \right)}{\sum_{i=1}^{N} t_i(T)}, \tag{8.6}$$

where $\forall j$, $t_j(T)$ is the timestamp achieved by node v_j at time T, and $\mathrm{dec}_l(i,j)$ is a logic variable identifying the pieces of information currently decoded by node v_l. In particular, $dec_l(i,j) = 1$ if v_l has recovered the generation $i \leq t_j(T)$ of node v_j, i.e. if $x_{v_j}^{t_j=i}$ is known by v_l at time T, and equals 0 otherwise. Since $\sum_{i=1}^{N} t_i(T)$ sums to the overall number of data items disseminated in the network from the beginning of the simulation the metric in (8.6) represents the percentage of overall information collected by v_l at a given instant of time. As an index of the performance of the whole network, we use

$$P(T) = \frac{\sum_{l \in A(T)} p^l(T)}{|A(T)|}, \tag{8.7}$$

that is the average value of the previous index computed on the set of nodes $A(T)$ that are active, i.e. in the on state at time T. In the case of a stable network with a constant information rate, $P(T)$ can be used to calculate the recovery time defined in Section 8.3. Indeed, the recovery time can be computed as $\{min(T)|P(T) = 1\}$, i.e. the first instant of time when all the nodes of the network know the overall information. Finally, we recall that all the numerical results based on the previous definitions have been averaged over 30 independent trials in order to compute a confidence interval and guarantee statistically meaningful values. Every trial uses a different seed for the random generators, thus resulting in statistically independent outcomes for both the network topology and the behavior of the nodes. We start comparing the coded (C) and uncoded (U) approaches in the simplest configuration, characterized by a stable network of $N = 1000$ nodes with $N_{neigh} = 50$ and constant information. In this case every node has to collect $N - 1$ pieces of information owned by the other nodes. This particular case allows us to validate the analytical model presented in



(a) U system (b) C system

Figure 8.4: Experimental recovery time and analytical model for the coded and uncoded systems as a function of m (stable network with constant information).

Section 8.3. In Fig. 8.4 we compare the experimental average recovery time for 5 values of information size $m = 100, 500, 1000, 2000, 3000$ bits and compare them with the analytical model of equation (8.4). The experimental results are shown as error bars reporting the 95% confidence interval whereas the full and dash lines represent the exact and approximated version of equation (8.4), respectively. First of all, it can be observed that the analytical model is quite accurate in the C case (right graph), with the exception of the smallest values of m. As anticipated the analytical results represent a lower bound for the performance of the U system (left graph). This is due to the coupon collector approximation adopted for the U model, that assumes that all data $x_{v_l}^{t_l}$ can be gathered by a certain node with equal probability. This assumption is clearly violated for large values of m, since a piece of information is soon shifted out of the packet body to make room for new information. In other words, every piece of information takes very few hops in this case. Under this circumstance, node v_l is not able to sample the pieces of information randomly and the coupon collector approximation does not hold. Another way to explain this phenomena is that the number of hops performed by a given piece of information is less than the mixing time of the graph. In conclusion this first set of experimental results on the one hand validates our analytical model. On the other hand, it confirms the significant advantage of C in terms of recovery time. As already noted the gap between the two approaches increases for larger information m, or equivalently when decreasing the maximum transfer unit (DIM). As an example,

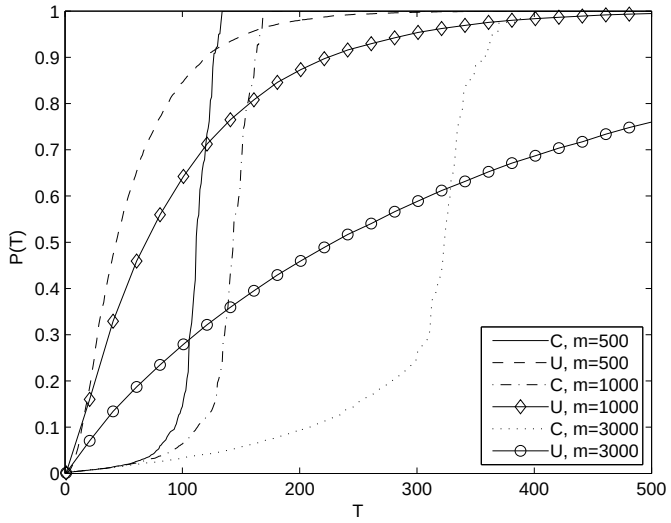


Figure 8.5: $P(T)$ as a function of time for some values of m (stable network with constant information)

for $m = 2000$ bits, the U solution exhibits a recovery time of 1653 ± 48 hops whereas the C solution achieves the same results in 237 ± 12 hops.

For the same scenario, in Fig.8.5 the value of $P(T)$, i.e. the average percentage of collected information, is shown for various values of m. Looking at this index, observe how the information propagates into the network as a function of time. The U system initially collects a great number of innovative pieces of information, but takes a long time to find the last bit of information. On the contrary, the C system, that is initially penalized by the observation of coded messages that do not allow it to know the actual information, is able to recover the whole information much earlier.

In the following, we investigate some system configurations characterized by a dynamic network topology and asynchronous information updates. In this case it is not possible to define the full recovery time. As a consequence, in this case our analysis will be based on the simulator results, and in particular on the behavior of the index $P(T)$.

The first case that we are interested in is that of a stable network where the nodes can change their information. In this scenario, we suppose that every t_c time slots n_c randomly chosen nodes update their information. The system performance clearly depends on the rate of the information update $r_c = \frac{t_c}{n_c}$: if the information changes very frequently it is not possible to retrieve all the information on time unless we increase

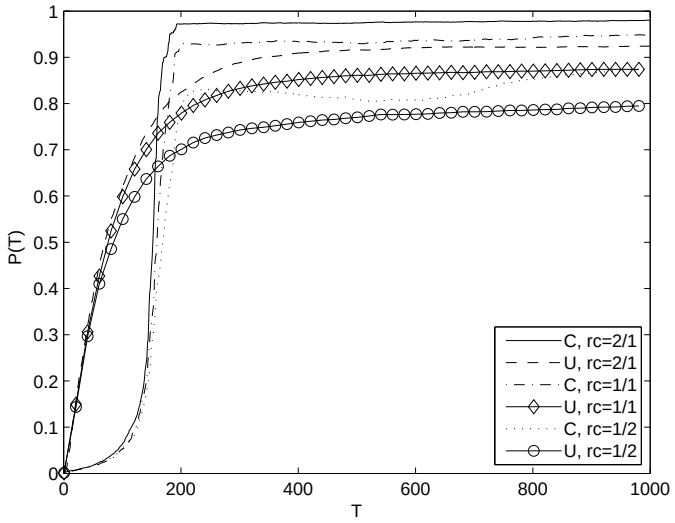


Figure 8.6: $P(T)$ for different r_c, with $N = 1000$ nodes, $N_{neigh} = 50$ and $m = 1000$ (stable network and dynamic information)

the network traffic, i.e. the value of w. In Fig. 8.6 $P(T)$ is shown for various values of the rate r_c, in a network of $N = 1000$ nodes, $N_{neigh} = 50$ and $m = 1000$. Note that the C system is more robust to the information updates than the U one for all the investigated rates r_c. As an example, in the case $r_c = 2/1$, i.e. every 2 time slots a node changes the information, the C system approaches a 100% recovery of the information in about 200 time slots; on the contrary, the U counterpart not only takes almost double the time to saturate to its maximum performance, but the achieved recovery is limited to slightly more than 90%.

Let us now investigate the effect of network dynamics. In this case we create a $G(V, E)$ with $N = 1000$ nodes broadcasting their local information. A randomly chosen set of i_l nodes is started in the off state; every t_l time slots n_l nodes pass from the off to the on state, i.e. they join the overlay, and the same number of nodes do the opposite, thus keeping constant the number of active nodes $|A(T)| = N - i_l$. In all the following simulations we let the rate of network dynamics, defined as $r_l = \frac{t_l}{n_l}$, vary and we fix $i_l = 5n_l$. In Fig. 8.8, 8.7 $P(T)$ is reported, showing that the proposed system is also robust to network dynamics. Note that when $r_l = 25/20$ and $m = 2000$, the recovery times for C and U are about 500 and 1000 hops, whereas increasing the network dynamic to $r_l = 50/20$ we get 750 and 1250 hops, respectively. As in all scenarios, the advantage

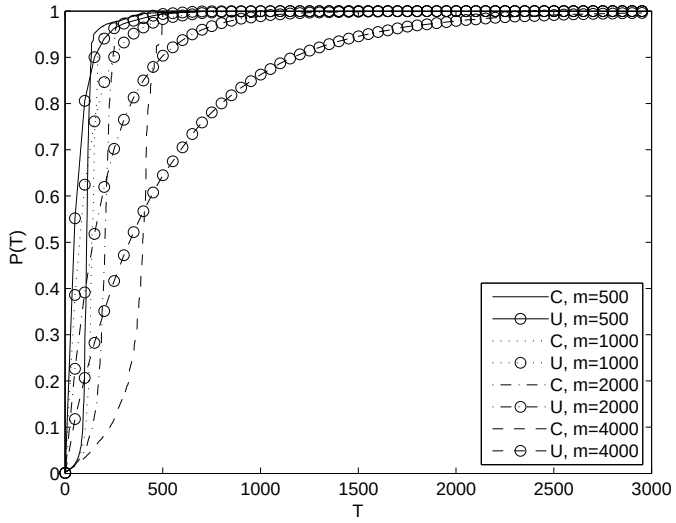


Figure 8.7: $P(T)$ for various values of m and $r_l = \frac{25}{20}$ with $N = 1000$, $N_{neigh} = 50$ (dynamic network and constant information)

.

of the C approach is more significant for larger m.

Finally, in Fig. 8.9-8.10 network dynamics and asynchronous information updates are considered jointly. In figure 8.9 $P(T)$ is shown for the case of $m = 2000$ for various ratios r_l and fixing the information change rate to $r_c = \frac{50}{20}$. As expected, the more the information in the network changes, the more the system has difficulty in retrieving the complete information. However, the C system consistently outperforms the U system. In figure 8.10, $r_c = \frac{50}{20}$ and $r_l = \frac{1}{1}$ are fixed, and we let m vary. As already noted the gain of C over U increases for larger values of m.

8.5 Other approaches

In the following we argue that alternative approaches to allow nodes of a distributed application to obtain a local view of some global system property are less effective and efficient.

The simplest solution to this problem is to have nodes report their local data to a well-known central repository node. Access to the global system property defined on all local data is obtained by sending queries to the repository that provides the requester

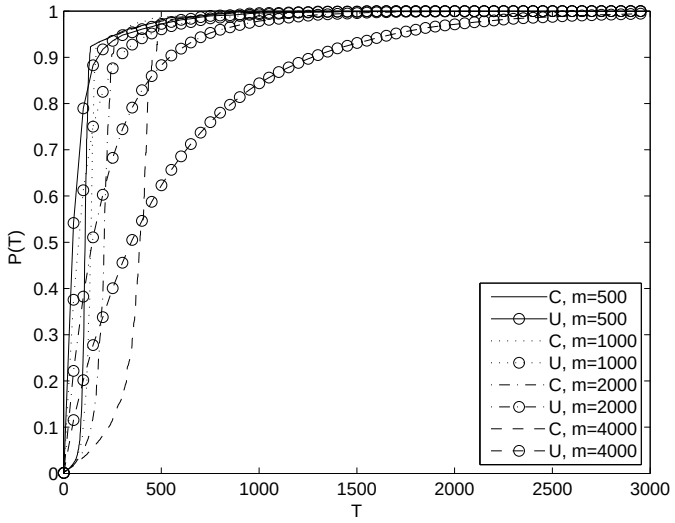


Figure 8.8: $P(T)$ for various values of m and $r_l = \frac{50}{20}$ with $N = 1000$, $N_{neigh} = 50$ (dynamic network and constant information)

.

node with one or more responses. This solution is neither scalable nor resilient: the aggregate request load is equal to $\lambda \cdot N$ and if we view the repository node as a server whose service capacity is equal to μ responses/sec, we obtain a load factor $\rho = \frac{\lambda \cdot N}{\mu}$ that quickly becomes greater than 1 leading to loss of queries and unbounded delays. A similar analysis must be done for the load generated by asynchronous data updates: the load factor for this service is $\rho_c = \frac{\lambda_c \cdot N}{\mu}$. Using the notation adopted in our simulations we have $\lambda_c = \frac{1}{N r_c}$.

Decentralized solutions might work better. Each node may spread its locally updated information by means of flooding or gossipping. In flooding, each node sends update messages to its neighbors containing the local data and the current generation number. These neighbors propagate the update messages to their neighbors up to a maximum number of hops (based on the message Time-to-Live). In gossipping, activity is organized in *rounds*: each node stores a maximum number of update messages. Each message is forwarded a maximum number of times, and each time a node forwards a message it randomly selects a subset of recipient nodes (the number of selected recipients is called the *fanout*). To spread the current generation version of the local data, a node starts a gossipping round. It is proved that atomic reliable broadcasting, i.e., all

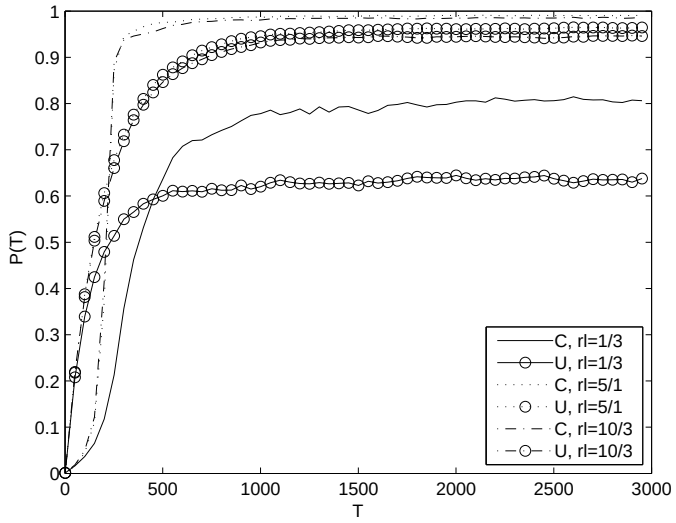
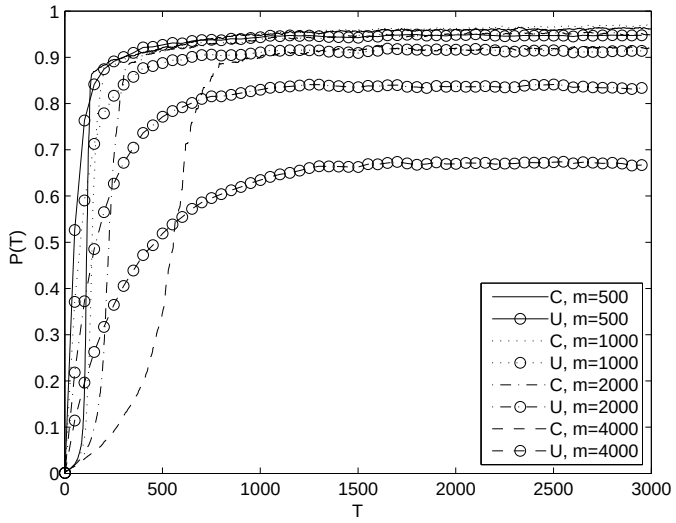


Figure 8.9: $P(T)$ in the case $m = 2000$, $r_c = \frac{50}{20}$ for various values of r_l (Dynamic network and information)

nodes receive the data disseminated by a round initiator, is achieved with high probability if the fanout is on average $O(logN)$ and taking $O(logN)$ rounds to complete (45). Flooding and gossipping introduce high redundancy, i.e., the same update message can be received several times by the same node, especially by those with a large number of incoming connections. This translates into the possibility of nodes saturating their processing and communication capacities. Indeed, it is easy to show that in the best case the load factor at each peer is $\rho_c = \frac{\lambda_c \cdot N}{\mu}$, the same as the centralized solution. Furthermore, this and other issues in gossipping were discussed in (47) where the authors explicit numerous hidden assumptions that are necessary to ensure robustness of gossip-based protocols. These assumptions thus make gossipping not a viable option in the context we consider.

8.6 Conclusion

In this chapter we have shown that the recent advances in rateless coding and decoding can be profitably exploited to achieve a robust and timely distribution of local information to all participants of a distributed application. A major feature of the proposed approach lies in the fact that we are able to exploit the network coding principle in



hh

Figure 8.10: $P(T)$ in the case $r_l = \frac{1}{1}$ and $r_c = \frac{50}{20}$ for various values of m (Dynamic network and information)

a scenario where local data can be updated asynchronously. Moreover as opposed to some forms of distributed storage proposed in the literature, our proposal realizes a continuous update of the global information across the whole distributed system while keeping the amount of traffic under control.

From the algorithmic point of view, the major contribution is the design of a novel decoder for rateless codes that is robust to asynchronous updates of the information. Another result is that we developed a simple analytical model for the estimation of the time required to spread the information as function of the network and information sizes, given a constraint on the MTU allowed by the available transmission protocol. The model can be exploited for the estimation of the performance and for the selection of important parameters of the system. The analytical analysis shows that the proposed coded approach reduces the time required to communicate all the information with respect to an analogous system without coding. Furthermore we prove that such a reduction increases with the size of the information to be spread, or analogously when the MTU shall be very limited. Another paramount result is that the coded system scales better than the uncoded one when the number of nodes increases in the distributed system. The simulator shows that the system is very efficient in many scenarios characterized by network dynamics and information updates that cannot be

analyzed with an analytical approach.

8. BROADCAST OF MULTIPLE SOURCES OF LOCAL INFORMATION IN DISTRIBUTED SYSTEMS

Chapter 9

Distributed Virtual Disks for Cloud Computing

9.1 Introduction to Cloud Computing

Cloud Computing applications, especially data intensive ones, typically need storage services that provide the illusion of infinite storage whose lifetime is not bound to that of the virtual machines (VMs) that use them. In many cases, applications or system-specific functions require the availability of raw block devices, for instance when a specific file system is necessary or when the application needs to directly access physical storage (e.g., in the case of a Database Management System).

Virtualized Block Devices (VBD) (like the Amazon Elastic Block Store (EBS) (65), the Eucalypts Block Storage Service (BSS), and the Virtual Block Store System (VBS) (66)) provide the abstraction of persistent block storage devices (henceforth referred to as *virtual disks*), whose lifetime is independent from the VM they are attached to, and whose size can be dynamically extended at anytime.

In general, VBD systems are implemented by coupling a back-end system that provides physical storage (typically a Networked Attached Storage (NAS) device) with a front-end that provides mechanisms and protocols to access and manage virtual disks. NAS-based solutions, however, are appropriate when the resources of the Cloud are located in a single datacenter, but may prove inadequate when storage resources are spread across different datacenters, or when the virtual disks must be accessible from the resources of an InterCloud (67) (i.e., a Cloud composed by a set of independent Clouds). In these

situations, the following issues arise:

- *Data availability*: if the datacenter where the NAS is located becomes unaccessible (i.e., because of a network component failure), the storage devices it provides to the VBD system become unavailable to externally-located users or applications;
- *Data confidentiality*: usually, the NAS keeps data private by means of encryption. That, however, introduces potentially very high computation overheads that may adversely affect the performance of a virtual disk;
- *Remote access performance*: if a given virtual disk must be accessed from a virtual machine located outside the datacenter where the corresponding NAS resides, performance is usually limited by the single network path used to transport data.

In order to cope with the above issues, we propose *ENIGMA* (84), a distributed infrastructure that abstracts the storage resources provided by a set of physical nodes, and exposes a set of virtual disks that can be used either directly by the individual virtual machines hosted on a Cloud infrastructure or as a back-end for VBD systems. ENIGMA is designed to provide a set of features tailored to Cloud Computing platforms, namely large storage capacity, high availability, strong confidentiality, high data access performance, and the ability to tune all these characteristics to the specific needs of the user or of the application, even at run-time.
ENIGMA achieves these goals by exploiting erasure-coding techniques, where each sector of a virtual disk is encoded as a set of n *fragments* independently stored on a set of physical storage nodes, and a minimum number k of fragments ($k \leq n$) is required to reconstruct a given sector.
Data confidentiality is ensured by encoding each sector in such a way that the number of fragments k needed to reconstruct the sector is large, and by keeping private both the coding function and the addresses of the storage nodes where fragments are stored. Instead data availability is ensured by properly choosing the total number n of fragments per sector, and by properly spreading them over the storage nodes. Data access performance is achieved by simultaneously fetching many fragments belonging on the same sector from the corresponding storage nodes, and by dynamically increasing or decreasing the value of n for specific sectors.

In general, cloud storage systems can be classified according to the interface they provide to users, applications and virtual machines. *Cloud file systems* (e.g., the Google File System (68) and the Hadoop Distributed File System (69)) provide a file system interface, while *digital object stores*, either targeted to Cloud systems (e.g., Amazon's S3 (70), Sector (72), Comet (73), and Depot (71)) or to general-purpose uses (e.g., Oceanstore (74, 75), Total Recall (76) and Storage@Home/Storage@Desk (77, 78)), provide a key-value interface in which generic "digital objects (*values*) are associated with a key, and are stored and retrieved by using that key. Compared to these systems, ENIGMA is placed at a lower abstraction layer since it provides a virtual disk abstraction.

Virtualized Block Device systems (e.g., Amazon's Elastic Block Store (EBS) (65), the Eucalypts Block Storage Service (BSS), and the Virtual Block Store System (VBS) (66)) provide a standard disk interface instead, i.e. they expose individual disk sectors to the operating system. These systems are more focused on providing higher-level functionalities like snapshotting, user authentication and storage sizing, while ENIGMA, by providing the abstraction of a virtual disk, complements them and can be used as their back-end in place of the traditional NAS-based solutions they usually rely upon.

In addition to ENIGMA, other systems provide the abstraction of virtual disks, namely *Petal* (80), *FAB* (79), and *Parallax* (81). These systems, unlike ENIGMA, are able to aggregate only storage resources located in the same data center, and do not provide specific mechanisms to treat the issues arising in geographically-distributed storage systems (as opposed to ENIGMA).

In addition to the differences outlined above, ENIGMA is (to the best of our knowledge) the only storage system providing virtual disks that directly incorporate confidentiality-ensuring mechanisms. Privacy-ensuring mechanisms are also provided by *Pasis* (82), a distributed storage system that uses a threshold scheme for coding to ensure availability and confidentiality, but at the cost of larger storage overhead than ENIGMA and without providing a disk-like interface.

9.2 Architecture

As already anticipated in the Introduction, ENIGMA provides the abstraction of a virtual disk by aggregating a set of geographically sparse storage resources. Virtual

disks consisting of a sequence of sectors, independently addressable, that are suitably encoded and stored on storage nodes in order to achieve availability, confidentiality, and performance. ENIGMA uses erasure codes, in particular rateless codes, to achieve its design goals.

The architecture of ENIGMA, schematically shown in Fig. 9.1, provides for two distinct logical entities, namely the *storage node* (that provides access to its local storage to store sector fragments) and the *proxy* (that coordinates the usage of storage nodes to provide virtualized disks). In an ENIGMA implementation, there may exist at any given time several proxies that use a set of storage nodes. A given virtual disk, however, is handled exclusively by a single proxy. To ensure scalability, storage nodes are organized into a

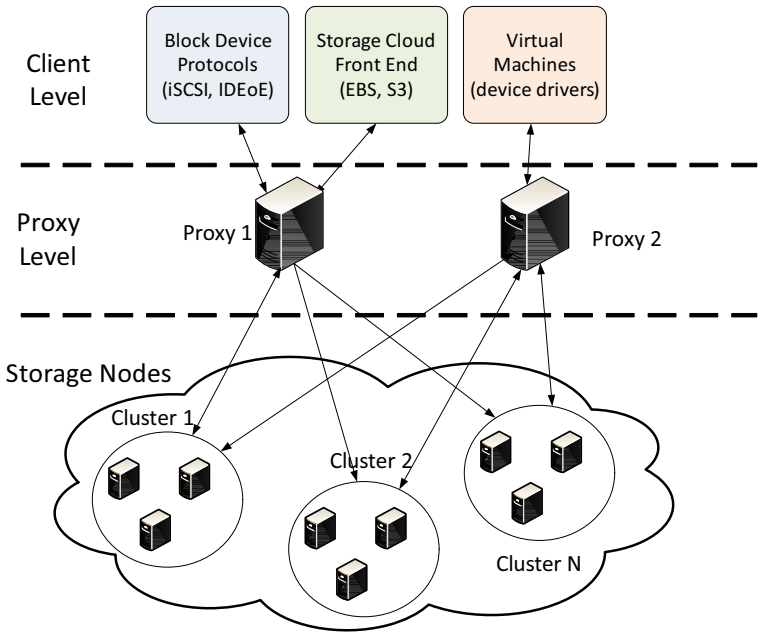


Figure 9.1: System architecture

set of *clusters*, each one coordinated by a *cluster head*. The proxy stores the individual fragments of each sector by sending each one of them to two or more cluster heads (for redundancy), and each cluster head autonomously decides which of the storage nodes it controls will actually store the fragment.

In the following, we first describe sector encoding, and then discuss the proxy level and the storage node.

9.2.1 Sector Encoding

Erasure coding techniques work by splitting a given sequence of bytes into a set of independent *fragments* in such a way that, in order to reconstruct the sequence, only a subset of these fragments are required.
In ENIGMA, coding is applied to the sectors of a virtual disk. More specifically, the encoding of a virtual disk VD, composed of m consecutive sectors s_i of identical size, is performed as follows. Each sector s_i is first divided in k fragments $s_i^1, \ldots, s_i^k$ and, subsequently by applying LT codes (9), it is transformed into a sequence of n fragments $c_{i,l}$, $l = 1, 2, \ldots, n$ with $n > k$, computed as an exclusive-or (XOR) of the fragments of sector s_i (which amounts to linear coding in the Galois Field of size 2, GF(2)) as:

$$c_{i,l} = b_{i,l} \cdot s_i = \sum_{j=1}^{k} b_{i,l}^j s_i^j,$$

where $b_{i,l} = b_{i,l}^1, \ldots, b_{i,l}^k \in \mathrm{GF}(2)^k$ is called the *bitmap* of the encoded fragment $c_{i,l}$. In practice, the bitmap is a random vector of length k in which each element is a 0 or 1. The number of 1s of a bitmap is called its *degree*. According to the relaxed reconstruction property of LT codes, any $K = k(1 + \epsilon)$ fragments taken from the n coded fragments can reconstruct the original sector. Thus up to $n - K$ failures of the nodes can be tolerated. The bitmaps are produced sequentially by a standard random generator initialized with a given *seed* z_i. The random outcomes leading to the bitmap $b_{i,l}$ can be reproduced by knowing z_i and the sequential number of the coded fragment l. The couple (z_i, l) can be interpreted as a key, required to correctly interpret and decode the coded fragment. On the Fly Gaussian Elimination (OFG) (15) decoding algorithm is used in order to decode the fragments.
Sector encoding provides the basic mechanism for simultaneously achieving availability, confidentiality, and performance.

- *Availability*: the availability of a virtual disk can be set to a given level by properly choosing the values of k and n; moreover, the availability can be increased using a particular redundancy increase policy.

- *Confidentiality*: storage nodes are unable to reconstruct any sector since (a) the coding seed z_i of a sector is unknown, and (b) far less than K coded fragments are stored per each node.

- *Performance*: the fragments of a given sector being stored on different storage nodes can be retrieved in parallel, thus reducing the time required to retrieve a sector compared with respect to the case in which the whole sector is stored on a single node.

9.2.2 The Proxy

The *Proxy* provides each client with access to a virtual disk that can be used by that client only, and manages the redundancy of its sectors in order to satisfy the availability, confidentiality, and performance requirements set for that disk.
In order to ensure data confidentiality, all the encoding and decoding operations for individual sectors are performed on the proxy, where the encoding seed of each sector is securely stored. We postulate that the proxy is placed on the premises of each client, that is responsible for its security. Furthermore, within the storage network, each sector s_i of a virtual disk VD is anonymously identified by means of its *hash_id* $h(s_i)$, which is computed as function of the pair (VD, s_i) in such a way that it cannot be inverted to determine the VD and s_i, so that an attacker who successfully decodes a sector does not know to which other sectors it relates (and thus the attacker is not able to reconstruct the entire disk).
When a sector s_i is written for the first time, the proxy determines the set $ch_{i1}, \ldots, ch_{in}$ of cluster heads in charge of storing its encoded fragments, and the proxystores this information (together with $h(s_i)$ and its encoding seed z_i) into a table, whose structure is shown in Table 9.1, plus anyadditional information needed for its operation. The

Table 9.1: Proxy table for virtual disk VD

sector_id	**hash_id**	**cluster_list**	**CRC**	**seed**
1	$hash\_id_1$	$ch_{11}, ch_{12}, \ldots$	CRC_1	z_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	$hash\_id_m$	$ch_{m1}, ch_{m2,,\ldots}$	CRC_m	z_m

CRC field in the proxy table stores a cyclic redundancy code (CRC) used to identify possible inconsistencies in the sector data, that might be caused (for instance) by data pollution attempts. More specifically, when a sector is written, its CRC is computed and stored by the proxy before it is encoded. When a sector is read, after it has been

decoded, the proxy recomputes its CRC proxy and compares it with the one stored before the write operation: if the two values coincide, the proxy deduces that no data corruption occurred (due to either normal events or malicious actions).
Upon receiving a read or write request for a given sector s_i of virtual disk VD, the proxy computes $h(s_i)$ and, by looking up the proxy table for that disk, determines the addresses of the heads of the clusters where s_i's fragments have been stored.
When the proxy receives a write request for sector s_i, it encodes s_i and sends the resulting fragments to the cluster heads stored in the corresponding row of the proxy table. Each fragment $c_{i,l}$ of s_i is stored as a tuple $< h(s_i), l, c_{i,l} >$, where l represents the sequential identifier of the coded fragment and $c_{i,l}$ is the actual data. The sector content can be retrieved only by the proxy from any K tuples $< h\_id, l, c_{i,l} >$. For each fragment of s_i the proxy chooses at random, with uniform probability, two or more clusters that will store it. In this way we avoid that a failure of a single cluster head compromises the entire sector. If the sector is *empty* (that is, it has never been written since the creation of the virtual disk), then the proxy considers the write operation to be complete. If, instead, the sector is not empty (that is, itcontains data that must be overwritten as result of the write operation), then all the fragments of s_i that are stored on the storage nodes must be overwritten before the write operation can be considered complete. If this were not done, a read operation for sector s_i could overlap with a write of the same sector and thus could lead to consistency problems in case some of the old fragments are retrieved and combined with some new ones. To avoid this problem, the proxy keeps the just-written sector s_i a local cache until the write operation is terminated (that is until all the fragments of s_i have been committed on the respective clusters), and sends to each cluster head an *invalidate* command together with the list of fragments assigned to that cluster head. Upon receiving this message, each cluster head eliminates the old fragments, stores the new ones and, when these operations have been completed for all the fragments, sends an acknowledgment back to the proxy.
When the proxy receives a read request for sector s_i, it sends a read request to every cluster head in charge of holding the fragments of s_i. The cluster head, by using the *fragment retrieval protocol* (described in the next section), forwards the request to all the storage nodes of the cluster that actually store fragments associated with $h(s_i)$. When the proxy has received K fragments, it decodes s_i and discards all the additional

fragments received after the CRC has been successfully computed.

In order to improve performance, the proxy provides for *overlapped operations*, where an operation can be performed as soon as its request arrives, without having to wait for the completion of another operation that has been issued previously. This behavior is different from the one exhibited by physical hard disks, where the presence of mechanical components that may need to be repositioned each time a new sector must be accessed imposes the completion of a pending request before a new one can be performed.

In addition to the basic read and write operations, the proxy dynamically manages the redundancy of selected sectors by increasing the redundancy of the most requested sectors (in order to reduce their retrieval time). In a dual way, it decreases the redundancy of sectors that are rarely used. In any case, the redundancy value of a given sector never goes below the threshold n that is set, for the whole virtual disk, to a value that guarantees a given availability level. This dynamic modification of sector redundancy also provides the basic mechanism that can be used to keep virtual disk performance at a given level in the face of migration of a Virtual Machine accessing it: the relocation of its sector can be performed by first expanding its redundancy and by placing the corresponding extra fragments in places "closer" to the new location of the Virtual Machine, and then by bringing back the redundancy to its original value and removing fragments located "far away".

Redundancy increase is accomplished by the proxy in the following way. After choosing which sector s_i has to undergo a redundancy increase, the proxy sends an *increase* command to two of the cluster heads storing fragments of s_i. These two cluster heads combine and code the fragments in their possession (as detailed in Sec. 9.2.3), and then send the newly generated fragments to another cluster head that will store them in its storage nodes. In this way, redundancy is increased without requiring the intervention of the proxy, that has only to update the table of the corresponding virtual disk in order to update the list of cluster heads managing s_i.

Redundancy reduction for sector s_i is instead performed by having the proxy ask every cluster in charge of handling fragments of s_i to delete a given number of fragments (determined by the proxy) sufficient to bring redundancy back to its standard value n.

9.2.3 The Storage Nodes

The *Storage Nodes* are a federation of heterogeneous machines, distributed geographically and possibly belonging to different organizations. To accommodate a potentially unlimited number of storage nodes, the storage nodes are organized into a hierarchical overlay network in which nodes are grouped into clusters (each node belongs to a single cluster). Each one is coordinated by a cluster head that is responsible for orchestrating across the nodes in the corresponding cluster the execution of the operations issued by the proxy. Cluster heads form a peer-to-peer network whose topology is irrelevant for the purposes of ENIGMA. Many of the overlay networks published in the literature (59)fit our needs.
Clusters and cluster heads are in charge of off-loading some of the work from the proxy, and of reducing the burden of managing the storage infrastructure. More specifically, as discussed below, the clusters take care of the diffusion and retrieval of the sector fragments, and of the increase and decrease of sector redundancy:

- *Fragment diffusion*: when a virtual disk VD is created, each one of its sectors is assigned to a specific cluster head. When sector s_i must be stored, the proxy sends to each one of the cluster heads responsible for s_i some of s_i's fragments. If sector s_i is written for the first time, the cluster head places the fragments on suitable storage nodes picked at random among the components of the cluster. Conversely, if s_i has been already written, the message sent by the proxy contains both an *invalidate* command and the new fragments to be stored in the cluster. In this case, before storing the new fragments, the old ones are deleted. After these operations have been completed, an acknowledgment is sent to the proxy that, as already discussed, can then discard the locally-cached copy.

- *Fragment retrieval*: upon receiving a retrieval request for a specific sector s_i (i.e., for the corresponding $h(s_i)$) value, the cluster head sends a broadcast message to all the members of its cluster requesting all the fragments associated with $h(s_i)$. Each storage node in that cluster reacts to that message by directly sending to the proxy all the fragments associated with $h(s_i)$.

- *Redundancy increase*: when the proxy decides to increase the redundancy of sector s_i, it sends a message $increase(h(s_i)$ to two cluster heads ch_x and ch_y. These

cluster heads are asked to combine their fragments associated with $h(s_i)$ and store them in a third cluster (which could also be one of the former two), with a standard write operation. Either one of ch_x and ch_y could act as a leader and start the coding protocol. In order to enable individual nodes to recode fragments without exposing to them the encoding seed z_i (to preserve confidentiality), and without requesting the intervention of the proxy (in order to avoid a potential performance bottleneck), a second level of coding is used. A given storage node may increase the redundancy of sector s_i by *recombining*, by means of pairwise XOR operations, a subset of fragments $c_{i,l}$ of sector s_i locally stored. The recombination of two fragments $c_{i,l1}, c_{i,l2}$ of s_i corresponds to creating a new combined fragment $c_{i,l1,l2}$ by XORing them, i.e. $c_{i,l1,l2} = c_{i,l1} \oplus c_{i,l2}$. The new fragment $c_{i,l1,l2}$ is stored as a tuple $< h(s_i), l1, l2, c_{i,l1,l2} >$. The recombined fragments can be used by the proxy to decode the sector content; its bitmap is the result of the XOR of the bitmaps of packets $l1$ and $l2$. When the redundancy increase operation is finished, the leading cluster head sends a message to the proxy containing the identifier of the cluster head responsible for storing the new fragments. The proxy updates the entry in the disk table for that sector.

- *Redundancy decrease*: the redundancy decrease command is sent by the proxy to each cluster head in the list for that particular sector s_i. Upon receiving the command $decrease(h(s_i))$, only the cluster heads that store recombined fragments perform the operations. They send a broadcast message to all the storage nodes in their cluster asking them to delete the recombined fragments of the sector. Each cluster head that deleted some fragments sends a message to the proxy that updates the disk table.

Another duty of the overlay of storage nodes is to periodically check the status of its participants. This could be used by the proxy as a keep alive function that is necessary in order to monitor the availability of sectors. The proxy periodically sends a *keep alive* message to each cluster head that stores fragments of a particular sector and the proxy evaluates the responses. The cluster heads that receive the keep alive message monitor in turn the storage nodes of their cluster. When storage node availability falls below a guaranteed threshold, the proxy can issue an increase redundancy command, restoring the original availability. If the number of original $c_{i,l}$ fragments decreases too

much, the second level of coding would not suffice to reconstruct the sector (recall that it encodes only a subset of fragments) and the original redundancy has to be restored. This operation is performed every time a write occurs. If write operations are scarce the proxy periodically, takes care of collecting the fragments and reconstructing the sector, coding it again and spreading the fragments.

9.3 Data Confidentiality Assessment

In ENIGMA the confidentiality of the data is guaranteed by three major means, namely the fragment dispersion, the disk addressing policies and the linear coding of fragments. According to the distributed nature of ENIGMA, the list of clusters hosting a given sector is not available in the clear and is secured in the proxy. Therefore both the cluster head and storage nodes cannot retrieve all the coded fragments of a sector. At the addressing level, each sector is identified through the anonymous *hash_id*, that is meaningful only for the proxy owning the disk. In this way, cluster and storage nodes are neither able to associate a coded fragment to a given VD nor do they know the sector sequential order.

Note the confidentiality related to the dispersion and addressing policies is not robust to malicious nodes able to mimic the proxy requests. In fact, an attacker could conceive of a malicious node implementation that sniffs the protocol signalling from a proxy, tries to reconstruct the association between the *hash_id* and (VD, s_i), and then attempts to retrieve at least K coded fragments by flooding the cluster head nodes with read requests. Nonetheless, this weak level of confidentiality is complemented by the high degree of security provided by the usage of the LT codes as detailed in the following.

The use of LT codes provides two levels of confidentiality. A first level of confidentiality is guaranteed by the fact that every storage node hosts a limited amount of coded fragments from which it is very unlikely one can reconstruct the sector content, even assuming that all the private information stored in the corresponding proxy has been hijacked. We refer to this kind of attack as a *single storage node attack*. The second form of protection that we analyze is related to the confidentiality of the coding seed z_i that prevents a malicious node able to collect at least K coded fragments from gaining any knowledge of the sector content. We call this attempt to violate the confidentiality

of the data as a *sector read attack*.

9.3.1 Single storage node attack

In this case the attacker is a modified storage node that attempts decoding the fragments of a sector she holds for local storage. In this case, we can show that ENIGMA is very secure even in the unlikely case that the attacker has come into possession of the coding seed z_i of the sector under attack.

Checking the *hash_id*s of the fragments, an attacker is able to collect all the fragments that belong to a certain sector, even if she is not able to know which sector that is. To decode the fragments, the attacker needs to have the bitmaps linked to the fragments. As already seen, these bitmaps are collected (and encoded) in the proxy table: storage nodes have no information about the bitmaps of the fragments they store. Assume that a modified storage node is able to obtain such information. Even if she knows the bitmaps of the stored fragments, the attacker still can not decode the fragments to obtain the sector. In fact, the number of fragments of the same sector (i.e. fragments labeled with the same *hash_id*) stored in a single storage node are not sufficient to perform a complete decoding: at least K fragments are needed to decode, but each storage node contains far less than K fragments of the same sector.

It is possible, however, to attempt to decode at least the fragments owned by a storage node. The process of decoding fountains codes with an insufficient number of fragments is called *partial decoding* (17). As already seen in Chapter 3, in a partial decoding process the decoder attempts to decode the maximum quantity of information using the limited number of owned fragments. The OPD algorithm (34) should be used to perform partial decoding. Fig. 9.2 shows the results of the partial decoding of LT codes for k in the range (64,1024). The percentage of decoded fragments is reported as a function of the percentage of coded fragments owned by the node. Both percentages are calculated with respect to k. In Fig. 9.2 we can see that if a (modified) storage node owns a number of fragments far smaller than k, she is not able to decode a large number of fragments even if she knows their linked bitmaps.

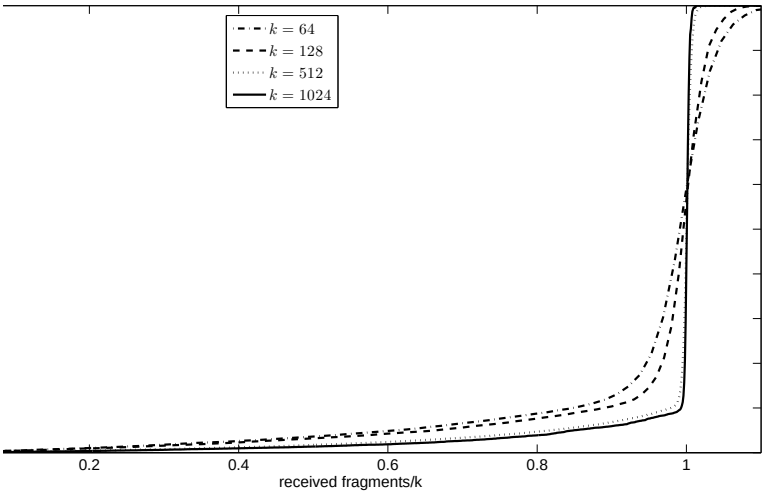


Figure 9.2: Percentage of decoded fragments vs. percentage of owned fragments.

9.3.2 Sector read attack

In the sector read attack, we assume that a malicious node has gained sufficient knowledge of the distributed storage system to retrieve a set of K coded fragments of the sector s_i. As already pointed out, such fragments allow the owner of s_i to run the LT decoder and reconstruct the sector content. This is possible because the owner has access to the seed z_i that corresponds to the knowledge of the K bitmaps $b_{i,l}$, $l = 1, \ldots, K$. The attacker, on the contrary, has no chance to decode the sector in absence of the bitmap. Indeed, the attacker could perform a brute force search among all the possible combinations of the bitmaps. For each combination, LT decoding can be attempted, but the attacker has no means of recognizing if the obtained sector is the correct one, making the attack impracticable.

This brute force attack turns out to be conceivable only if the CRC of the sector, retained by the proxy, has been leaked. In this case, the attacker can detect the correct sector by means of the CRC. The large number of required decoding attempts, however, makes the attack unfeasible. Indeed an exhaustive search amounts to testing all possible configurations of K bitmaps, each composed of k bit strings. In the most general case, the number of trials required turns out to be K^{2^k}. Since we can assume that the attacker knows the degree distribution $P(d = k) = \mu(k)$ used for LT coding, a smarter

search can be designed. Instead of testing all possible bitmaps, the attacker would limit the search to the most likely outcomes of the bitmap degree. The sequence constituted by K outcomes of the bitmap degree can be seen as independent and identically distributed random variables $d_1, d_2, \ldots, d_K$. According to the *asymptotic equipartition property* (60) we can define the typical set of degree outcomes. It is well known that the typical set has a probability close to 1, all elements of the typical set are nearly equiprobable, and the number of elements in the typical set is nearly $2^{KH(\mu)}$, where $H(\mu)$ is the entropy associated to the RSD. From the point of view of the attacker, the typical set represents the set of the most likely degree sequences to be tested in order break the code. Since the RSD is a peaked distribution where only a limited set of the possible k degrees have non negligible probability, $H(\mu)$ is typically small. As an example, setting the parameters of the RSD $(c, \delta) = (0.01, 0.001)$ (9) one obtains $H(\mu) \approx 3$. It turns out that the typical degree sequences are approximatively 2^{3K}. For each degree sequence there are $\prod_{j=1}^{K} \binom{k}{d_j}$ possible combinations of the K bitmaps to be tested. Taking into account the properties of the binomial and that the most likely degree yielded by the RSD is $d = 2$ we can use the following lower bound on the number of combinations

$$\prod_{j=1}^{K} \binom{k}{d_j} \gg \prod_{j=1}^{K} \binom{k}{2} = \left(\frac{k(k-1)}{2} \right)^K$$

We thus conclude that the number of decoding attempts that an attacker needs to perform to break the code is larger than

$$2^{3K} \left(\frac{k(k-1)}{2} \right)^K \tag{9.1}$$

with $K = k + \epsilon$. Clearly, the exponential complexity of the search makes it practically unfeasible. As an example in Fig. 9.3 the lower bound of Equation (9.1) is reported as a function of k in the range $(64, 2048)$.

9.4 Availability Evaluation

In this section we evaluate the availability level that ENIGMA provides for its virtual disks by first devising an analytical model of availability, and then using it to quantitatively compute the availability for specific values of n and k. We assume that each

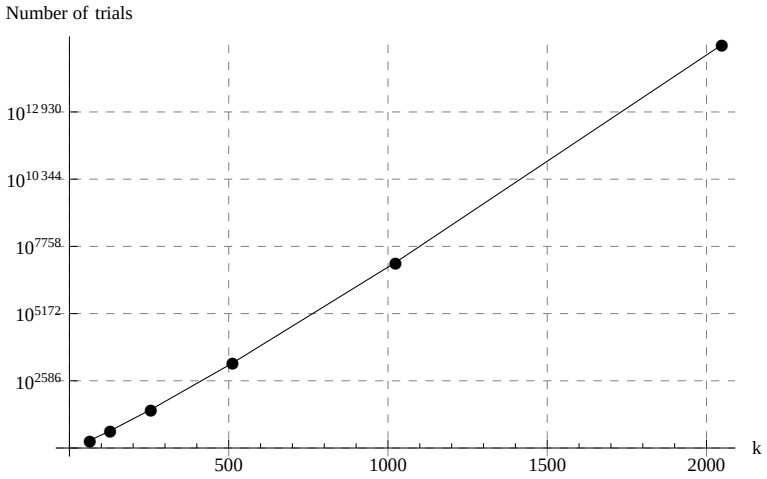


Figure 9.3: Lower bound on the trials required to break the LT code as a function of k for RSD with $(c, \delta) = (0.01, 0.001)$.

sector is split into k fragments, that are subsequently encoded as n fragments, that are stored on n independent nodes. Consequently, when a proxy wants to recover a sector, it asks all the n storage nodes to forward the fragments of that sector. We also assume that all the storage nodes are characterized by the same reliability p (i.e., the probability of successfully providing a sector fragment to the requesting proxy).
Under these assumptions, the probability $p(r)$ for the proxy to receive r encoded fragments is given by

$$p(r) = \binom{n}{k} p^k (1-p)^{n-k}.$$

As already seen in earlier sections, the proxy is able to decode the fragments (i.e. to recover the sector) only if $r \geq k$. In particular, if the proxy receives $r \geq k$ fragments it has a probability $\sigma(r)$ to decode the fragments. This probability depends on the rateless code and the decoding algorithm used: in (28) a theoretical study of $\sigma(r)$ for LT codes and a Gaussian Elimination-like decoding algorithm is presented. Finally, using the law of total probability, it is possible to calculate the probability that the proxy decodes the requested sector, i.e. the availability, as

$$P_k = \sum_{i=k}^{n} p(i) \cdot \sigma(i).$$

In Tab.9.2 P_k is evaluated as a function of the storage node's reliability p for several values of n and for some values of the code block length $k = 64, 128, 256$. As expected,

Table 9.2: P_k as a function of p for several values of k and n.

	$k = 64$			$k = 128$			$k = 256$		
p	P_k			P_k			P_k		
	$n = 83$	$n = 96$	$n = 128$	$n = 166$	$n = 192$	$n = 256$	$n = 333$	$n = 384$	$n = 512$
0.70	0.015	0.302	0.932	0.004	0.500	0.993	0.009	0.923	1.000
0.75	0.091	0.570	0.964	0.085	0.845	0.998	0.325	0.996	1.000
0.80	0.294	0.760	0.981	0.462	0.945	0.999	0.919	0.999	1.000
0.85	0.558	0.860	0.989	0.827	0.977	0.999	0.993	1.000	1.000
0.90	0.741	0.915	0.995	0.931	0.990	1.000	0.998	1.000	1.000
0.95	0.829	0.941	0.996	0.967	0.995	1.000	1.000	1.000	1.000

the availability increases if more coded fragments are stored in the system, i.e. if one increases the storage overhead. In Tab.9.2 the values of n corresponding to a storage overhead of 30%, 50% and 100% respectively have been selected. Tab.9.2 also shows the dependency of P_k on the value of k. It can be observed that, given a certain value of p, the required level of availability can be guaranteed for increasing k, i.e. cutting each sector into more fragments. This amounts to using an LT code with a larger block size and therefore closer to the channel capacity bound.

The above considerations allow one to derive the basic rules for the fulfillment of a service level agreement that, in the most general case, can be varied on a per sector basis. This great flexibility is empowered by the usage of rateless codes where the sector fragmentation k and the number of coded fragments stored in the system n can be changed adaptively, e.g. in response to the migration of a virtual machine using the cloud disk.

9.5 Performance Evaluation

In order to show that ENIGMA is able to provide adequate performance, we performed a simulation study (carried out by means of a discrete-event simulator we developed) in which we compared the time taken by a client to retrieve a sector when using, respectively, ENIGMA and a virtual disk located on a single storage server (the *baseline system*).

In our experiments, we consider a virtual disk consisting of about 30 million 40 KB

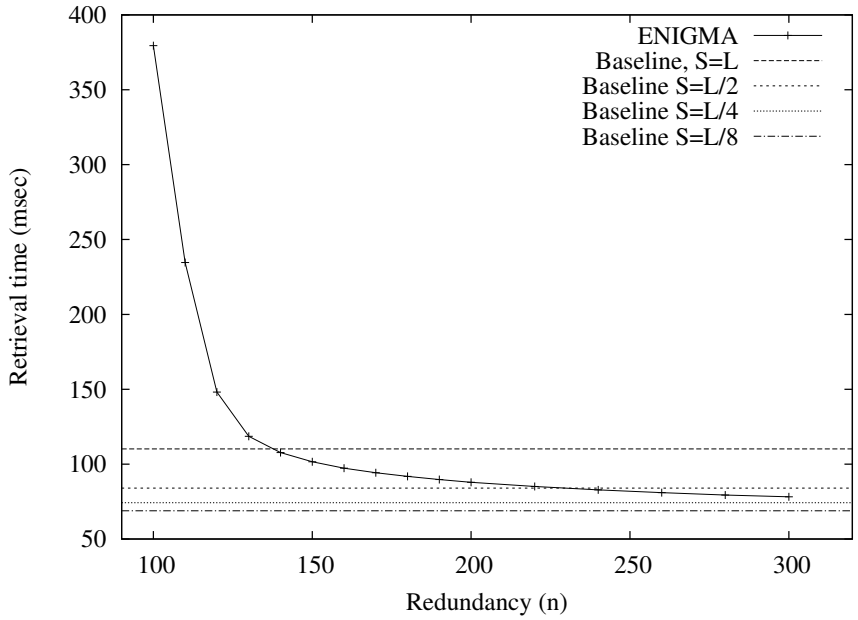


Figure 9.4: Comparison of ENIGMA performance w.r.t. the baseline system.

sectors. We considered an ENIGMA configuration consisting of 2000 perfectly reliable storage nodes, whose proxy-to-node end-to-end latencies were extracted from the data published by the *Meridian* project (61); the average latency L of the ENIGMA configuration was $L = 32$ msec. For each sector, we set $k = 100$, and we performed several experiments in which we increased n from k to $3k$ with a size step of 0.1. The resulting fragments were randomly distributed, with uniform probability, over the 2000 storage nodes. The workload used in our experiments consisted of a stream of sector requests contained in a trace, downloadable from the *SNIA trace repository* (62), and described in (63).

The results obtained for the scenarios involving ENIGMA have been compared with those obtained by the baseline system for different values of its *service time* S (the time taken to serve a single sector request). In particular, we performed experiments for $S \in L, L/2, L/4, L/8$.

The results of our experiments are reported in Fig. 9.4 (for all of them, we computed 95% confidence intervals with 2.5% relative error). As can be observed from this figure, the larger n, the smaller the sector retrieval time. This is not unexpected, as a sector is considered to be retrieved when $k + \epsilon$ fragments out of n are received, and the larger n,

the larger the probability that the required fragments are shipped by the faster nodes. From Fig. 9.4, we can also observe that:

- for $n > 140$ ENIGMA performs better than the baseline system when $S = L$ (i.e., $S = 32$ msec. in our case);
- for $n > 220$ ENIGMA performs better than the baseline system when $S = L/2$ (i.e., $S = 16$ msec);
- for $n \sim 300$ ENIGMA performance are very close to that of the baseline system when $S = L/4$ (i.e. $S = 8$ msec.).

These results can be considered promising, given that typical access time of virtualized disks has been measured to be in the range $15 - 20$ msec (64). Furthermore, we expect that, by adopting proper caching strategies (based on redundancy increase and decrease) that are part of our future work, performance should significantly increase, thus making ENIGMA competitive for smaller values of n.

9.6 Conclusions and Future Works

ENIGMA is a distributed infrastructure that provides virtual disks that can be used either directly by the VMs hosted on a Cloud infrastructure, or as the back-end for VBD systems. We exploited rateless coding techniques to encode each sector of a virtual disk as a set of fragments independently stored on a set of physical storage nodes to achieve tunable large storage capacity, high availability, strong confidentiality, and high data access performance.

We described the ENIGMA architecture based on proxy nodes that coordinate the usage of storage nodes providing access to their local storage to store encoded sector fragments. Storage nodes are organized in clusters and each cluster is coordinated by a cluster-head. The set of cluster-heads forms a logical peer-to-peer network with arbitrary topology.

We demonstrated that ENIGMA is resilient to attacks by a single malicious storage node attempting to decode fragments of disk sectors as well as by a more sophisticated attacker that has gained sufficient knowledge of the distributed storage system to retrieve a set of K coded fragments of a particular sector. Furthermore, we developed

analytical models to derive design criteria to obtain a desired availability level and we also discussed how to obtain high data access performance by letting the proxy manage overlapped operations, by simultaneously fetching many fragments from the same sector, and by dynamically increasing or decreasing sectors redundancy (this last feature also provides the basic mechanism that can be used to keep virtual disk performance at a given level in the face of migration of the VM accessing it). The performance results we obtained via simulation, albeit preliminary, can be considered encouraging.
Future work includes the development of suitable fragment distribution policies (in place of the random one adopted in this paper), of sector caching strategies, and of fragment migration techniques supporting the migration of the VM accessing the corresponding virtual disk. In particular, ENIGMA can be used as a framework for Rateless Regenerating Codes, presented in Chap. 5.

References

[1] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. *Informed content delivery across adaptive overlay networks*. IEEE/ACM Trans. on Networking, 12(5):767780, Oct. 2004. 1, 59, 60

[2] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. *Bullet: high bandwidth data dissemination using overlay mesh*. In 19th ACM SOSP, pages 282 297, Oct. 2003. 1, 59, 60

[3] M. Wang and B. Li. *R2: Random push with random network coding in live peer-to-peer streaming*. IEEE Journal on Selected Areas in Communications, 25(9):16551666, Dec. 2007. 1, 59, 75

[4] C. Wu and B. Li. *rStream: resilient and optimal peer-to-peer streaming with rateless codes*. IEEE Trans. on Parallel and Distributed Systems, 19(1):7792, Jan. 2008. 1, 29, 59, 61, 64, 69, 71, 76

[5] C. Fragouli, J. Widmer, and L. B. J. *Efficient broadcasting using network coding*. IEEE/ACM Trans. on Networking, 16(2):450462, Apr. 2008. 1, 59

[6] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. *XORs in the air: practical wireless network coding*. IEEE/ACM Trans. on Networking, 16(3):497510, June 2008. 1, 59, 75

[7] C. Gkantsidis and P. Rodriguez. *Network coding for large scale content distribution*. In IEEE INFOCOM 2005, Mar. 2005. 1

[8] R. Koetter and M. Medard. *An algebraic approach to network coding*. IEEE/ACM Trans. on Networking, 11(5):728795, Oct. 2003. 1, 59

[9] M. Luby. *LT codes*. In IEEE FOCS, pages 271280, Nov. 2002. 1, 2, 16, 17, 53, 60, 76, 93, 94, 96, 100, 119, 128

[10] D. MacKay. *Fountain codes*. IEEE Proc. Communications, 152(6):10621068, Dec. 2005. v, 1, 15, 52, 53

[11] C. Studholme and I. Blake. *Windowed Erasure Codes*. The IEEE International Symposium on Information Theory, pp.509-513, 2006. 2, 19, 48

[12] A. Kamra, V. Misra, J. Feldman and D. Rubenstein. *Growth Codes: Maximizing sensor network data persistence*. SIGCOMM Comput. Commun. Rev., vol. 36, no. 4, pp.255-266, 2006. 2, 31, 91

[13] A. Shokrollahvi. *Raptor Codes*. IEEE Transactions on Information Theory, vol.52, pp. 2551-2567, June 2006. 2, 19, 76

[14] K. Hu, J. Castura and Y. Mao. *Performance-Complexity Tradeoffs of Raptor Codes over Gaussian Channels*. IEEE Communication Letters, vol. 11, no. 4, April 2007. 3

[15] V. Bioglio, M. Grangetto, R. Gaeta and M. Sereno. *On the fly Gaussian Elimination for LT codes*. IEEE Communication Letters, vol. 13, no. 12, December 2009. 4, 78, 119

[16] Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Services (MBMS); Protocols and Codecs (Release 6), 3rd Generation Partnership Project (3GPP), Technical Report 3GPP TS 26.346 v6.3.0, 3GPP, 2005.

[17] S. Shangavi. *Intermediate performance of rateless codes*. Information Theory Workshop, 2007. v, 4, 30, 31, 126

[18] B. Krishnamurthy, C. Wills and Y. Zhang. *On the use and performance of content distribution networks*. In IMW 01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, pages 169182, New York, NY, USA, 2001. ACM. 5

[19] G. Pallis and A. Vakali. *Insight and perspectives for content delivery networks*. Commun. ACM, 49(1):101106, 2006. 5

[20] A. M. K. Pathan and R. Buyya. *A taxonomy and survey of cdns*. Technical Report GRIDS-TR-2007-4, The University of Melbourne, Australia, Feb 2007. 5

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

[22] Y. Wu, A. Dimakis, and K. Ramchandran. *Deterministic regenerating codes for distributed storage*. in Proc. 45th Allerton Conf. Comm. Control and Computing, Sept. 2007. 5, 7

[23] A. G. Dimakis, P. B. Godfrey, Y.Wu, M.Wainwright, and K. Ramchandran. *Network coding for distributed storage systems*. Infocom 2007, 2007. 7, 51, 52, 56

[24] T.C. Jepson. *The basics of reliable distributed storage networks*. IT Professional, 6(3):1824, May-June 2004. 7

[25] S. Kim, K. Ko, S. Chung. *Incremental gaussian elimination decoding of raptor codes over BEC*. IEEE Communications Letters, 12(4):307-309, April 2008. 21, 22, 25, 95, 96

[26] K. Hu, J. Castura, Mao Y. *Performance-complexity tradeoffs of Raptor codes over gaussian channels*. IEEE Communications Letters, 11(4):343-345, April 2007. 21, 29

[27] A. Talari, N. Rahnavard. *Rateless Codes with Optimum Intermediate Performance*. GLOBECOM 2009. 31

[28] S. Kim, S. Lee. *Improved Intermediate Performance of Rateless Codes*. ICACT, 3:1682-1686, February 2009. 32, 129

[29] F. Lu, C. H. Foh, J. Cai, L. T. Chia. *LT Codes Decoding: Design and Analysis*. ISIT 2009 30, 37, 46

[30] V. G. Subramanian, D. J. Leith. *On a Class of Optimal Rateless Codes*. Proc. Allerton Conference 2008

REFERENCES

[31] T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, J. Shi, B. Leong *A Random Linear Network Coding Approach to Multicast.* IEEE Trans. on Information Theory, 52(10):4413-4430, October 2006. 41, 59, 61

[32] P. Maymounkov, N. J. A.Harvey. *Methods for Efficient Network Coding.* Proc. Allerton Conference 2006 41

[33] J. Byers, M. Luby, M. Mitzenmacher, A. Rege. *A Digital Fountain Approach to Reliable Distribution of Bulk Data* Proc. ACM SIGCOMM98, Sept. 1998 13

[34] V. Bioglio, M. Grangetto, R. Gaeta and M. Sereno. *An Optimal Partial Decoding Algorithm for Rateless Codes* Proc. ISIT 2011, Aug. 2011 4, 95, 96, 98, 126

[35] C. Cooper. *On the Rank of Random Matrices* Random Structures and Algorithms 16(2):209-232, March 2000 48

[36] A. G. Dimakis, K. Ramchandran, Y. Wu and C. Suh. *A Survey on Network Codes for Distributed Storage* Proceedings of the IEEE 99(3):476-489, March 2011 52

[37] C. Gkantsidis, P. Rodriguez. *Network coding for large scale content distribution* IEEE INFOCOM 2005, March 2005 59

[38] S. Puducheri, J. Kliewer, T. E. Fuja. *The Design and Performance of Distributed LT Codes* IEEE Trans. on Information Theory 53(10):3740-3754, October 2007 3, 60, 76

[39] R. Gummadi, R. S. Sreenivas. *Relaying a fountain code across multiple nodes* IEEE ITW08, May 2008 61

[40] M. Wang, B. Li. *How Practical is Network Coding?* 14th IEEE IWQoS, June 2006 61, 62, 91

[41] M. Grangetto, R. Gaeta, M. Sereno. *Rateless codes network coding for simple and efficient P2P video streaming* IEEE ICME 2009 64, 69, 71

[42] M. E. J. Newman, S. H. Strogatz, D. J. Watts. *Random graphs with arbitrary degree distributions and their applications* Physical Review E 64, 2011 69

[43] S. Spoto, R. Gaeta, M. Grangetto, M. Sereno. *Analysis of PPLive through active and passive measurements* HotP2P 2009 69

[44] http://www.pplive.com 69

[45] A. M. Kermarrec, L. Massoulié, A. Ganesh. *Probabilistic reliable dissemination in large-scale systems* IEEE Transactions on Parallel and Distributed Systems 14(3):248-258, March 2003 89, 91, 111

[46] M. Jelasity, A. Montresor, O. Babaoglu. *Gossip-based aggregation in large dynamic networks* ACM Transactions on Computer Systems 23(3):219-252, August 2005 89, 91

[47] L. Alvisi and J. Doumen and R. Guerraoui and B. Koldehofe and H. Li and R. van Renesse and G. Tredan. *How robust are gossip-based communication protocols?* Operating Systems Review 41(5):14-18 October 2007 91, 111

[48] S. Deb, M. Medard, C. Choute. *Algebraic gossip: a network coding approach to optimal multiple rumor mongering* IEEE Transactions on Information Theory 52(6):2486-2507 June 2006 89, 91, 93

[49] A. G. Dimakis, V. Probhakaran, K. Ramchandran. *Distributed Fountain Codes for Networked Storage* IEEE ICASSP 2006 3, 91

[50] Y. Lin, B. Liang, B. Li. *Data Persistence in Large-scale Sensor Networks with Decentralized Fountain Codes* IEEE Infocom 2007 3, 91

[51] D. Vukobratovic, C. Stefanovic, V. Crnojevic, F. Chiti, R. Fantacci. *Rateless Packet Approach for Data Gathering in Wireless Sensor Networks* Selected Areas in Communications, IEEE Journal on 28(9):1169-1179, September 2010 91, 92, 93

[52] J. Haupt, W.U. Bajwa, M. Rabbat, R. Nowak. *Compressed Sensing for Networked Data* IEEE Signal Processing Magazine 25(2):92-101, March 2008 92

[53] R. Gaeta, M. Grangetto, M. Sereno. *Local Access to Sparse and Large Global Information in P2P Networks: A Case for Compressive Sensing* Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on, August 2010 92

[54] C. Fragouli, J. Le Boudec, J. Widmer. *Network coding: an instant primer* Computer Communication Review, 36(1):63-73, January 2006 75

[55] P. A. Chou, Y. Wu, K. Jain. *Practical network coding* Proceedings of the Annual Allerton Conference on Communication Control and Computing, 41(1):40-49, 2003 75, 78

[56] J. Heide, M. V. Pedersen, F. H. Fitzek, T. Larsen. *Cautious view on network coding-from theory to practice* Journal of Communications and Networks (JCN) 10(4):403-411, 2008 76

[57] N. Cleju, N. Thomos, P. Frossard. *Selection of network coding nodes for minimal playback delay in streaming overlays* IEEE Transactions on Multimedia, 13(5)1103-1155, October 2011 77

[58] A. Fiandrotti, S. Zezza, E. Magli. *Complexity-Adaptive Random Network Coding for Peer-to-Peer Video Streaming* IEEE International Workshop on Multimedia Signal Processing (MMSP) 2011, October 2011 77, 83, 86

[59] S. Banerjee, B. Bhattacharjeey, C. Kommareddy. *Scalable application layer multicast* SIGCOMM02, October 2002 123

[60] T. M. Cover, J. A. Thomas. *Elements of Information Theory* Wiley, 1991 128

[61] B. Wong, A. Slivkins, E. Gun Sirer. *Meridian: A Lightweight Network Location Service without Virtual Coordinates* SIGCOMM Conference, August 2005 131

[62] *The SNIA Block I/O Traces* http://iotta.snia.org/traces/list/BlockIO, Retrieved on Nov. 22^{nd}, 2010 131

[63] S. Kavalanekar, B. Worthington, Qi Zhang, V. Sharda. *Characterization of storage workload traces from production Windows Servers* IEEE International Symposium on Workload Characterization, October 2008 131

[64] M.R. Ahmadi, D. Maleki. *Effect of Virtual Techniques in Data Storage Access* 24^{th} International Conference on Advanced Information Networking and Application Workshops (WAINA), April 2010 132

[65] *Amazon Elastic Block Storage* http://aws.amazon.com/ebs, Retrieved on Nov. 22^{nd}, 2010 115, 117

[66] X. Gao, M. Lowe, M. Pierce. *Supporting Cloud Computing with the Virtual Block Store System* 5^{th} IEEE International Conference on e-Science (e-Science '09), December 2009 115, 117

[67] R. Buyya, R. Ranjan, N. Calheiros. *InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services* 10^{th} International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), May 2010 115

[68] S. Ghemawat, H. Gobioff, S. T. Leung. *The Google File System* 19^{th} ACM Symposium on Operating Systems Principles, October 2003 117

[69] T. White. *Hadoop: The Definitive Guide* O'Reilly, 2009 117

[70] *The Amazon Simple Storage Service (Amazon S3)* http://aws.amazon.com/s3, "Retrieved on Nov. 22^{nd}, 2010 117

[71] P. Mahajan and S. Setty and S. Lee and A. Seehra and A. Clement and L. Alvisi and M. Dahlin and M. Walfish. *Depot: Cloud storage with minimal trust* USENIX Symposium on Operating Systems Design and Implementation OSDI 2010, October 2010 117

[72] Y. Gu, R. L. Grossman. *Sector: A high performance wide area community data storage and sharing system* Future Generation Comp. Syst., 26(5):720-728, 2010 117

[73] R. Geambasu, A. Levy, T. Kohno, A. Krishnamurthy, H. M. Levy. *Comet: An active distributed key/value store* OSDI, 2010 117

[74] http://oceanstore.cs.berkeley.edu/ 117

[75] S. Rhea and P. Eaton and D. Geels and H. Weatherspoon and B. Zhao and J. Kubiatowicz. *Pond: the oceanstore prototype* FAST '03 Proceedings of the 2nd USENIX Conference on File and Storage Technologies 2003 117

[76] R. B. Kiran, K. Tati, Y. Cheng, S. Savage, G. M. Voelker. *Total Recall: System Support for Automated Availability Management* NSDI, 2004 117

[77] A. L. Beberg, S. P. Vijay. *Storage@home: Petascale distributed storage* IPDPS, 2007 117

[78] H. H. Huang, J. F. Karpovich, A. S. Grimshaw. *Analyzing the feasibility of building a new mass storage system on distributed resources* Concurrency and Computation: Practice and Experience, 20(10):1131-1150, 2008 117

[79] Y. Saito, S. Frølund, A. Veitch, A. Merchant, S. Spence. *FAB: building distributed enterprise disk arrays from commodity components* SIGPLAN Not., 39(11):48-58, October 2004 117

[80] E. K. Lee, C. A. Thekkath. *Petal: distributed virtual disks* 7^{th} international conference on Architectural support for programming languages and operating systems, 1996 117

[81] A. Warfield, R. Ross, K. Fraser, C. Limpach, S. Hand. *Parallax: Managing Storage for a Million Machines* 10th Workshop on Hot Topics in Operating Systems, 2005 117

[82] G. R. Ganger and P. K. Khosla and M. Bakkaloglu and M. W. Bigrigg and R. Garth and S. Oguz and P. Vijay and C. A. N. Soules and J. D. Strunk and J. J. Wylie. *Survivable storage systems* DARPA Information Survivability Conference and Exposition, 2001 117

[83] R. Ahlswede, N. Cai, S.-Y. R. Li, R. W. Yeung. *Network Information Flow* IEEE Transactions on Information Theory, 46:1204-1216, 2000 41

[84] M. Zola and V. Bioglio and C. Anglano and R. Gaeta and M. Grangetto and M. Sereno. *ENIGMA: Distributed Virtual Disks for Cloud Computing* IEEE International Parallel and Distributed Processing Symposium, May 2011 116