Encoding for Efficient Data Distribution in Multi-hop Ad hoc Networks

Luciana Pelusi, Andrea Passarella, and Marco Conti
IIT-CNR, Via G. Moruzzi, 1 – 56124 Pisa, Italy
{firstname.lastname}@iit.cnr.it

1. Introduction

The diffusion of pervasive, sensor-rich, network-interconnected devices embedded in the environment is rapidly changing the Internet. The nature of pervasive devices makes wireless networks the easiest solution for their interconnection. Furthermore, in a pervasive computing environment, the infrastructure-based wireless communication model is often not adequate: it takes time to set up the infrastructure network, while the costs associated with installing infrastructure can be quite high [1]. Therefore multi-hop ad hoc wireless technologies represent one of the most promising directions to further extend the current Internet, and diffuse traditional networking services even more widely ([2], [3]).

The heterogeneity of devices to be interconnected (from small sensors and actuators to multimedia PDAs), and the large spectrum of communication requirements (from few meters coverage and few kilobits bandwidth to city-wide coverage and broadband communications), have produced a set of multi-hop ad hoc network technologies. On the one hand, we have sensor networks for communications among small sized, low cost, and low energy-consuming devices (sensors) for which a high data rate is not necessary [4]. On the other hand, we have mesh networks that, by exploiting an infrastructure of wireless mesh routers, guarantee the exchange of multimedia information among devices located inside an urban area [5]. The work in [6] provides a complete overview of the ad hoc networking techniques by presenting their principles and application scenarios, and pointing out the open research issues. Among the ad hoc networking techniques, *opportunistic networks* [7] represent the most interesting scenario for the application of the encoding techniques analyzed in this chapter. Opportunistic networks represent the evolution of the multi-hop ad hoc network concept in which the end-to-end connectivity constraint is released. Indeed, typically, mobile ad hoc networks rely on the end-to-end principle, i.e. a path must continuously exist between the sender and the receiver for successful communications. In reality, end-to-end connectivity is a strong requirement only for interactive services such as VoIP, gaming, video streaming; many other applications can still correctly operate relaxing the end-to-end constraint, e.g., data applications like messaging, e-mails, data sharing, etc. In principle they can operate even if a sender-receiver path never exists [7]. In opportunistic networks, a node stores the messages in its local memory until a suitable forwarding opportunity exists. In this way packets are not discarded during network disconnections but are locally stored. The communication is still multi-hop with intermediate nodes acting as routers that forward the messages addressed to other nodes but, in this case, forwarding is not “on-the-fly” since intermediate nodes store the messages when no forwarding opportunity exists (e.g., there are no other nodes in the transmission range, or neighbouring nodes are considered not useful to reach the destination), and exploit any contact opportunity with other mobile devices to forward information.

In all these scenarios, a big challenge is efficient data distribution. Applications like messaging, content distribution and peer-to-peer applications are becoming more and more widespread in the today Internet. Thanks to their decentralized features, they are likely to play a major role in ad hoc environments such as opportunistic networks. One of their major requirements are networking techniques to efficiently convey

This work was partially funded by the Information Society Technologies program of the European Commission under the HAGGLE and BIONETS FET-SAC projects.

data to possibly large sets of users. In the legacy wired Internet, researchers have proposed solutions based on IP multicast and, more successfully, on peer-to-peer overlay networks. While p2p solutions for wired networks are quite well established, designing similar systems to enable efficient data distribution over mesh, opportunistic and delay-tolerant networks is a big challenge still far to be satisfactorily addressed. In the field of sensor networks, content-distribution and p2p applications are not very likely. However, typical sensor network applications such as environmental monitoring require efficient data distribution techniques, too.

The very characteristics of wireless communications make legacy solutions for data distribution designed for the wired Internet not effective in ad hoc environments, even in static scenarios. Systems designed for ad hoc networks cannot assume that "bandwidth is for free", as most legacy systems do. In addition, they have to efficiently cope with sudden changes of the wireless links characteristics. Dynamic topology reconfigurations, due either to user mobility or to energy management techniques that temporarily switch off some nodes, add further dimensions to the problem. These constraints are peculiar of wireless multi-hop ad hoc networks, and legacy systems are typically not able to cope with them. Data distribution systems, either targeted to p2p and content distribution paradigms, or designed for sensor network applications, are thus today an exciting research area for multi-hop ad hoc networking environments. Encoding techniques are an important building block to address many issues that arise in these systems.

The main idea of encoding techniques applied to networking protocols is not to send plain data, but to combine (encode) blocks of data together, in such a way that coded blocks can be interchangeable at receivers. In the simplest example, a source node willing to send k packets actually encodes the k packets into n encoded packets, with $n>>k$. Encoding is performed so that a receiving node has not to exactly receive the k original packets. Instead, *any* set of k packets out of the n encoded packets generated at the source is sufficient to decode the k original packets. Different receivers might get different sets of encoded packets, and still be able to decode the same original information. These encoding algorithms, known as *erasure coding*, have been originally applied to implement efficient reliable multicast protocols for wired Internet [8]. Recently they have found interesting application scenarios in wireless ad hoc networks. For example, erasure code techniques have been used in [9] for the design of a reliable point-to-point protocol for data transmission over wireless sensor networks. Other applications of erasure code techniques over wireless sensor networks are reported in [10] and [11]. In [10] they have been exploited as a means to achieve minimum-energy data transmissions, while in [11] they have been used to support distributed data caching over the network and facilitate subsequent data retrieval. Erasure codes have also been used to support speech communications over well-connected mobile ad hoc networks [12] since they are able to reduce the total transfer delay experienced by data packets, and make it adequate to real-time applications.

In the case of multi-hop ad hoc networks erasure code techniques have been generalized by introducing several encoding phases of the same data. This generalization is known as *network coding*. With *network coding*, data packets are encoded both at source nodes, and also at intermediate hops in the path towards receivers. Assuming that the communication paradigm is one-to-many, the source node encodes locally generated data packets and sends them to a subset of current neighbors. Such intermediate nodes generate a new set of encoded data packets based on the received packets, and further disseminate them towards the final destinations. Network coding has shown to be a very promising solution for data dissemination in multi-hop ad hoc networks [13]. Actually, it is able to provide very high reliability and exploit bandwidth very efficiently. It has been successfully applied to achieve optimal energy consumption in multicast [14], unicast [15], and also broadcast [16] transmissions over mobile ad hoc networks. A lot of attention has also been devoted to possible applications of network coding over mesh networks [17] [18], as well as over opportunistic [19], and vehicular ad hoc networks [20].

In this chapter we provide a detailed overview of the research efforts on encoding techniques for multi-hop ad hoc networks. In Sections 2 and 3 we deeply describe the basic encoding techniques network coding is based on. Section 4 deals specifically with network coding, describing its operations and discussing its applications to multi-hop ad hoc networks. Finally, Section 5 draws conclusions and highlights open issues of this field.

2. Coding Theory: Motivation and Basic Concepts

Reliable transmission of data over *noisy channels* has been a major concern to the communication engineering for a long time and appropriate control systems as well as recovery mechanisms for corrupted data have been thoroughly studied. A real breakthrough in this field was achieved in 1948 when Claude

Shannon [21] demonstrated that by efficiently *coding* messages at the sender before transmission and conversely decoding (possibly corrupted) messages that arrive at the receiver, it is possible to repair the effects of a noisy channel. The decoding system acts on the corrupted messages and strives to reconstruct the original messages. Encoding of data is equivalent to adding some *redundancy* to transmission so as the encoded data which is transmitted, i.e., the *code*, stores the information that has originally been produced at the source more robustly. Indeed, redundancy is exploited at the receiver both to realize that the incoming code is corrupted and to correct it. In the first case the code is said to be *error detection code*, in the latter, *error correcting code*.

The principle behind error detecting and correcting codes is also embedded in everyday language. Each language, in fact, is characterized by a vocabulary composed by a certain number of words. Words are sequences of letters (symbols) belonging to a particular alphabet. Albeit the total number of words included in a vocabulary is very high, it is far less than the total number of possible combinations of the alphabet letters. Thank to this property, if a misprint occurs in a long word, it can easily be recognized because the word (likely) changes into something that does not correspond to any other word of the vocabulary and rather resembles the correct word more than it resembles any other known word.

Adding redundancy to the information to transmit is equivalent to *mapping* the set of all the possible pieces of information that a source can generate, hereafter referred to as *words*, into a far bigger information-set of so-called *code-words*. The mapping function is such that each word has a correspondent code-word associated to it whereas a code-word may not correspond to any significant word produced at the source. The mapping process is referred to as *encoding* process. At the destination an inverse mapping is performed named *decoding* to reconstruct the original word transmitted by the source. Fig. 1 shows the encoding, transmitting and decoding processes for the data originated at a source.

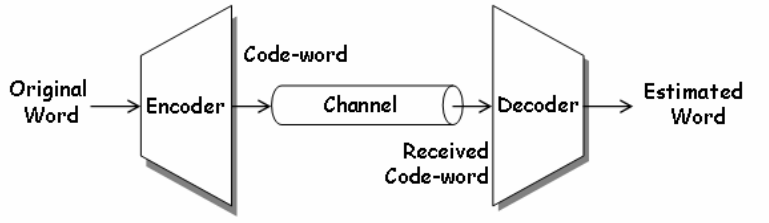

Fig. 1. Encoding, transmitting and decoding processes for the data originated at a source.

The decoding algorithms adopted at the receiver can be of different types. A *complete* decoding algorithm decodes every possibly received code-word (even corrupted) into a corresponding word. However, in some situations an *incomplete* decoding algorithm could be preferable, namely when a decoding error is very undesirable. When receiving new signals from the channel a new code-word should be read. The receiver infers each single symbol (bit) of the code-word from the signals received from the channel, so, for each received signal it has to decide whether it is much more similar to a 0-signal or to a 1-signal. Luckily, in most cases this decision is straightforward. Otherwise, it might be preferable to put a "?" instead of guessing whether the symbol is 0 or 1. In the subsequent decoding stage the receiver has to reconstruct the original word from the received code-word possibly including some "?"-s. An incomplete decoding algorithm *corrects* only those code-words which contain few errors whereas those with more errors cause decoding failures and give rise to so-called *erasures* in the sequence of the received words. The receiver can either ignore erasures or, if possible, ask for retransmission.

The success of an encoding/decoding system is strictly related to the characteristics of the channel over which transmissions occur and to the amount of redundancy which is introduced to transmission. Shannon formulated this relation in terms of *information rate* and *channel capacity*. He defined the *information rate* as the ratio between the number of significant words of a vocabulary and the total number of possible code-words that can be represented given a certain alphabet of symbols[1]. In addition, he defined the *channel capacity* as the total amount of information that a channel can transmit. Finally he demonstrated that, in case

[1] Given *code C* the set of code-words corresponding to a certain vocabulary and given *n* the number of bits over which code-words are represented, then the information rate (or just the rate) of the code *C* is defined as $R = \frac{\log_2 |C|}{n}$, where |*C*| gives the total number of code-words of the set *C*. Please, note that code *C* is only a sub-set of all the elements that can be represented over *n* bits and that each element of it matches a word of the vocabulary. Hence the cardinality of code *C* is equal to the cardinality of the vocabulary.

the information rate is lower than the channel capacity, there exist an efficient system for encoding and decoding such that the probability of corruption due to transmission is lower than ε, for each arbitrarily small $\varepsilon > 0$. This means that it is possible to transmit any information with an arbitrarily small probability of error by using the right encoding technique.

The code system proposed by Shannon, known as *Shannon code*, relies on a *probabilistic* approach and uses a random generation for code-words to assign to the words produced at the source. This choice is motivated by the fact that it is quite unlikely that a random process generates code-words similar to one another. The *similarity* between code-words is to be intended in terms of the *Hamming distance* that gives the total number of symbols that two code-words are differing in[2]. If for any couple of code-words generated by a coding system the code-words are sufficiently far from each other (differ in many symbols) then they are unlikely to be confused. Hence, whenever a code-word is sent over a channel and modified by noise, it is quite improbable that the corrupted code-word arriving to destination can be confused with another code-word. This would happen if only had the noise reduced the Hamming distance between the two code-words so much that they could be considered the same. If the initial Hamming distance is quite long, then this should not happen and the reconstruction of the original source word should be easy and unique.

In the Shannon coding system a source word is a string of *m* symbols (*m>0*) generated at the source site. These strings have to be encoded into strings of size *n* symbols i.e., code-words, with *n* higher than *m*, and subsequently sent. Choosing *n* higher than *m*, words generated at the source are mapped over a bigger space with a higher dimension (so the Hamming distance between code-words is likely to be high). The choice of the correct lengths, *m* and *n*, of both words and code-words can be made once targeted a specific $\varepsilon > 0$. Then an appropriate *encoding/decoding system* can be selected such that the probability not to reconstruct the original word sent by the source is lower than ε.

Unfortunately Shannon code cannot be implemented. Specifically, the decoding system cannot be implemented because it should evaluate the Hamming-vicinity of the received corrupted code-word to any other code-word of the code and select the code-word with the minimum distance as the un-corrupted code-word (this search has a dramatic cost) and then apply the inverse mapping to reconstruct the original word.

Many codes have been conceived after the original Shannon code. They have such algebraic characteristics that not only can the decoding process be implemented but it can also be very fast. Anyway, the *delay* caused by both encoding and decoding computations as well as the increase in the total transmission time due to the lengthening of the data to transmit, has been the main limiting factor to the diffusion of encoding techniques, and still remains their main drawback.

Error correcting codes have mainly been used, so far, in dedicated and critical communications systems (e.g., satellite systems) where they serve as a means to avoid expensive and long-lasting retransmissions. In these systems, encoding and decoding algorithms are typically implemented on dedicated hardware which allows overcoming the computational burden. Error correcting codes are also used in other special cases e.g., in modems, over wireless or otherwise noisy links, in order to make the residual error rate comparable to that of dedicated, wired connections. Error detection codes have mainly been used, instead, in the context of general purpose computer communications to discard the corrupted frames arriving from the communication channel, actually, they are quite easier to implement than error correcting codes. Error detection codes (see for example the Cyclic Redundancy Checksums (CRCs)) are managed at the lowest layers of the protocol stack (physical and data-link layers) that are typically HW-implemented in the network adaptor. While error correcting codes within streams of bits have seldom been implemented in these general purpose systems, especially in SW, correction of errors can be implemented quite more efficiently if focusing on missing packets rather than on corrupted symbols (i.e., bits). In fact, SW-implemented codes are actually being successfully developed in the last few years. They are known as *Erasure Codes* because they deal with erasures, i.e. missing packets in a stream. Erasure codes are implemented above the data-link layer where information is organized in *packets* rather than bit-streams (frames) and the channel noiseness is only perceived, after error processing and *detection* of the lower-layer protocols, through packet loss i.e., erasures. Erasures originate from errors that cannot be corrected at the data-link layer (but those are not frequent with properly designed and working hardware), or, more frequently, from congestion in the network which causes

[2] If $\underline{x} = (x_0, x_1, ..., x_{n-1})$ and $\underline{y} = (y_0, y_1, ..., y_{n-1})$ are two tuples, each one long *n* bits, then their Hamming distance is defined as follows. $d(\underline{x}, \underline{y}) = \left| \left\{ i \mid 0 \le i < n, x_i \ne y_i \right\} \right|$, where |.| is the cardinality of the set ".".

otherwise valid packets to be dropped due to lack of buffer space. Erasures are easier to deal with than errors since the exact position of the missing packets is known. Recently many efforts have been spent in the construction of *Erasure Codes*. They rely on the same concepts of the aforementioned error detecting and correcting codes but operate on packet-sized data objects rather than on bit-streams, and can be implemented in software using general purpose processors.

The motivation for recent deployment of high-layer erasure codes is slightly different from that of previous low-layer error codes. Erasure codes are attracting interest in the new emerging communication scenarios where existing protocols are becoming unsuitable, or at least inefficient, because for example, are based on frequent handshaking between the communicating peers, or because assume the existence of a continuous end-to-end connection between peers. Erasure codes have the potential to add to communication protocols i) reliability of transmissions, ii) scalability to the increasing number of hosts, iii) adaptability to the topology changes of a network (e.g., due to mobility or even power saving strategies), and iv) robustness to node/link failures. These are very promising features especially in scenarios like multicast transmissions, wireless networks (e.g., ad hoc, sensor, vehicular ad hoc, etc.), satellite networks, delay tolerant networks, and intermittently connected networks.

Software implementation for erasure codes is surely a challenging task due to the computational complexity. However, it has also been demonstrated that it can efficiently be done [22] [23] [8]. The rest of the chapter will focus on erasure codes and their more recent extension, network coding. In the following section some types of erasure codes will be presented, from the oldest to the most recent ones and some examples of applicative scenarios will be given for each of them, as well.

3. Erasure Codes

Erasure codes are generally considered implementations of so-called *Digital Fountains*. Digital Fountains are ideal, unlimited data-streams injected into the network by one or more source nodes. Nodes that are willing to receive information from those source nodes need to catch data from the stream until they have collected enough information to fulfil their needs. The way receivers catch data from the unlimited stream resembles the way people quench their thirst at a fountain. No matter which drops of water fall down on earth and which are drunk, the only important thing is that enough drops are caught. Source nodes generate digital fountains sending out the data they are willing to send and adding redundancy to it via encoding. The redundancy added is practically unlimited because the generated data flow is infinite. Namely, suppose that k data packets are originated at a source node. After encoding an unlimited stream of data packets is generated and injected into the network. The receiver node needs to receive *exactly* k data packets to be able to reconstruct the original source data. Moreover, *any* k data packets are suitable to the scope. Digital Fountains have been introduced in [24] and so far have mainly been used in conjunction with multicast protocols to improve reliability of transmissions.

Multicast protocols suffer from the so-called feedback implosion that occurs when multiple receivers ask for retransmission of data to the server. Multiple retransmissions are due to the inevitably different loss patterns that affect different receiver nodes. Clearly, it is not affordable to sending nodes to manage multiple retransmissions of different data packets towards different receivers. If the server generates a digital fountain instead, receiver nodes only have to receive an amount of data packets that corresponds to the exact number of original data packets. Indeed, *any* set of packets of size equal to the number of original data packets is appropriate. Hence, whenever receivers miss some packets it is sufficient that they wait for the subsequent redundant packets to come. No retransmissions are needed towards different receivers, no feedback channel is strictly necessary.

Content dissemination in environments with a vast number of receivers having heterogeneous characteristics can greatly benefit from this approach (see, for example, distribution of software, archived video, financial info, music, games, etc.). In these environments receivers will wish to access data at times of their choosing, they all will have their own access speeds and, in addition, their access times will probably overlap with one another. Besides addressing the retransmissions' issue, in this case digital fountains allow receivers to join the transmitted stream at any point and without requiring any synchronization among them. Synchronization is neither needed between senders and receivers.

Other challenging target scenarios for digital fountains are satellite networks where channel characteristics, i.e., high latency and limited as well as intermittent capacity, make retransmission-based solutions totally unworkable. Finally, wireless networks are emerging as a very promising scenario for digital fountains

because of the hardly unreliable communications and the common asymmetry between upstream and downstream channels. In addition, the recent deployment of opportunistic communication paradigms in the framework of the wireless networks characterized by intermittent connectivity looks quite an interesting application field, as well.

Although very promising, it should be noted that digital fountains are only an abstract concept. An unlimited stream of data is not feasible in practise, neither advisable, because it has the potential to flood and congest the network. During the last few years multiple approximations of digital fountains have been proposed trying to find a trade-off between efficacy and computational costs (see [25]-[30]). All these approximations can be expressed in terms of *(n, k)-codes*, where *k* gives the number of source data blocks[3] which are encoded while *n* gives the number of data blocks that the encoder produces from the original *k* blocks, and that are actually sent over the network. Fig. 2 shows the encoding and decoding processes of an (n, k)-code. During transmission over the network, some encoded data blocks may get lost due to congestion or corruption and only a sub-set of encoded blocks of size k', $k \le k' \le n$, arrives to destination and enters the decoder. The decoder gives back the original k source data blocks.

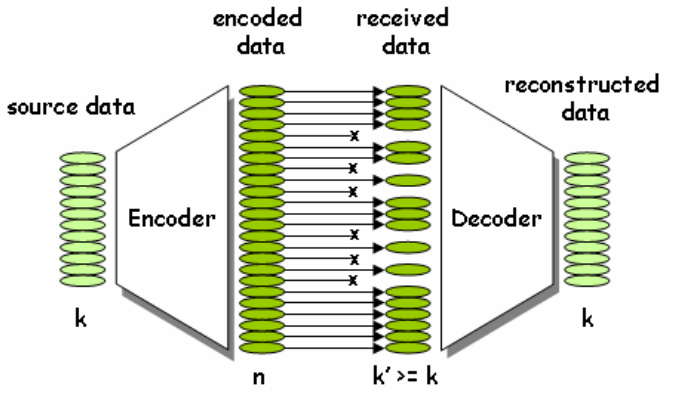

Fig. 2. A graphical representation of the encoding and decoding processes of an (n, k)-code.

As will be shown in detail in the next sections, to be able to reconstruct the source data blocks, the decoder needs at least *k* blocks of encoded data. Some implementations actually need to get at least *more* than *k* blocks to decode the original data. In the optimal case, an (n, k)-code allows the receiver to recover from up to *n-k* losses in a group of *n* encoded blocks.

In the remainder of this section insights on four types of erasure codes will be given: Reed-Solomon codes, Tornado codes, Luby-Transform codes, and Raptor codes. The order followed in the presentation is chronological since these codes have been conceived in a temporal sequence each one being an improvement of the previous one.

3.1. Reed-Solomon Codes

Reed-Solomon codes have been introduced in [25] as implementation of digital fountains. They are currently used to correct errors in many systems including wireless or mobile communications (e.g., cellular telephones, microwave links, etc.), satellite communications, digital television, and high-speed modems such as ADSL and xDSL. They are also used in storage devices (e.g., tape, Compact Disk, DVD, barcodes, etc.).

Assume a source data block is a word and let a sequence of *k* words be represented by a vector, say $\underline{x}$, of *k* elements. Encoding is represented by an encoding function *f(.)* which is applied to $\underline{x}$ and produces an encoded vector of *n* code-words. When the encoding function is *linear*, the code is said to be linear too. Reed-Solomon codes are a special case of linear codes as will be better explained later, after the following brief introduction to general linear codes.

[3] Here *block* is a generic piece of data. In the following sections *block* will be used to refer either a symbol or a packet or even a set of packets. Specific interpretations will be given when needed.

3.1.1. Linear Codes

Linear codes can be represented by a matrix G (encoding matrix) and encoding can be represented by a matrix-vector multiplication. Hence, a vector of code-words $\underline{y}$ corresponding to a vector of words $\underline{x}$ is simply given by the product $G \cdot \underline{x}$. Remembering the definition of an (n, k)-code, the encoding process looks as follows.

$$\underline{y} = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ \dots \\ \\ \dots \\ y_{n-1} \end{pmatrix} = G \cdot \underline{x} = \begin{pmatrix} g_{0,0} & g_{0,1} & \dots & \dots & g_{0,k-1} \\ g_{1,0} & g_{1,1} & \dots & \dots & g_{1,k-1} \\ \dots & \dots & & & \dots \\ \dots & \dots & & & \dots \\ \\ \dots & \dots & & & \dots \\ g_{n-1,0} & g_{n-1,1} & \dots & \dots & g_{n-1,k-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ \dots \\ x_{k-1} \end{pmatrix},$$

where $\underline{x} = (x_0, x_1, ..., x_{k-1})^T$ is the vector of *k* source words, $\underline{y} = (y_0, y_1, ..., y_{n-1})^T$ is the vector of *n* code-words, and $G_{(nxk)}$ is the encoding matrix. The encoding matrix must have rank *k*.

Suppose the destination node receives at least *k* (otherwise decoding is not possible) out of the *n* code-words produced by the encoder at the source, and let $\underline{y'}$ be a vector of *k* elements picked up among the received code-words. The encoding process that has generated this vector follows.

$$\underline{y'} = \begin{pmatrix} y_{i,0} \\ y_{j,1} \\ \dots \\ \dots \\ y_{l,k-1} \end{pmatrix} = G' \underline{x} = \begin{pmatrix} g_{i,0} & g_{i,1} & \dots & \dots & g_{i,k-1} \\ g_{j,0} & g_{j,1} & \dots & \dots & g_{j,k-1} \\ \dots & \dots & & & \dots \\ \dots & \dots & & & \dots \\ g_{l,0} & g_{l,1} & \dots & \dots & g_{l,k-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ \dots \\ x_{k-1} \end{pmatrix}.$$

The encoding matrix $G'_{(kxk)}$ is a *kxk* matrix obtained by extracting from $G_{(nxk)}$ those rows that correspond to the elements of the vector $\underline{y'}$. So, for example, if the *j*-th code-word of the original code-word vector (i.e., y_j) is inserted as the 2nd element in the vector $\underline{y'}$ (i.e., $y_{j,1}$), then the *j*-th row of the matrix $G_{(nxk)}$ is picked up and inserted as the 2nd row in the matrix $G'_{(kxk)}$.

Clearly, decoding means finding the solution to the linear equation $G' \cdot \underline{x} = \underline{y'}$, as follows.

$$\underline{x} = G'^{-1} \underline{y'}.$$

Note that the destination must be sure to identify the row in $G_{(nxk)}$ corresponding to any received element of $\underline{y}$. Note also that the set of rows corresponding to $\underline{y'}$ have to be linearly independent.

3.1.2. Encoding process of RS-codes

As has been introduced above, Reed-Solomon codes are a special case of linear codes. Source words are seen as coefficients of a polynomial of degree *k-1*, whereas code-words are seen as values of the polynomial worked out at *n* different points that can be chosen arbitrarily. Let the polynomial be as follows.

$$p(x) = a_0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + ... + a_{k-1} x^{k-1},$$

where $a_0, a_1, ..., a_{k-1}$ are the *k* words generated at the source for transmission and $p(x)$ is a single code-word obtained by evaluating the polynomial at the point x. The encoding process for a Reed-Solomon (n, k)-code is thus as follows.

$$\begin{pmatrix} p(x_0) \\ p(x_1) \\ p(x_2) \\ .. \\ .. \\ .. \\ p(x_{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 & .. & .. & x_0^{k-1} \\ 1 & x_1 & x_1^2 & x_1^3 & .. & .. & x_1^{k-1} \\ 1 & x_2 & x_2^2 & x_2^3 & .. & .. & x_2^{k-1} \\ .. & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. \\ 1 & x_{n-1} & x_{n-1}^2 & x_{n-1}^3 & .. & .. & x_{n-1}^{k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ .. \\ .. \\ .. \\ a_{k-1} \end{pmatrix},$$

where $x_0, x_1, ..., x_{n-1}$ are the n points selected for evaluation of the polynomial. They can be chosen arbitrarily, for example, for simplicity of encoding, or alternatively they can be *all* the possible integer values that can be represented over the number of bits available.

The encoding matrix of Reed-Solomon codes has a special form characterised by a geometric progression in each row. Such matrices are named *Vandermonde* matrices.

3.1.3. Decoding process of RS-codes

The decoding process consists in reconstructing all the polynomial coefficients $a_0, a_1, ..., a_{k-1}$ in a unique way. As is commonly known, for this to be possible, it is sufficient to know the value of the polynomial at exactly k points, i.e., receiving k code-words is sufficient. Hence, assuming that the identity (e.g., the sequence number) of the code-words that have arrived to destination is known, the coefficients of the polynomial can be derived at the destination site by solving the following system of equations.

$$\begin{pmatrix} y_{i,0} \\ y_{j,1} \\ .. \\ .. \\ y_{l,k-1} \end{pmatrix} = \begin{pmatrix} 1 & x_{i,0} & x_{i,0}^2 & .. & x_{i,0}^{k-1} \\ 1 & x_{j,1} & x_{j,1}^2 & .. & x_{j,1}^{k-1} \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ 1 & x_{l,k-1} & x_{l,k-1}^2 & .. & x_{l,k-1}^{k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ .. \\ .. \\ a_{k-1} \end{pmatrix}.$$

The matrix utilized in the decoding process is a sub-matrix of the encoding matrix and is obtained by selecting the k rows which correspond to the arrived code-words (in the example the i^{th}, j^{th} ..., and the l^{th} rows have been selected). The system admits a solution if the matrix is non singular. The determinant of the above *kxk* Vandermonde matrix corresponds to the following expression.

$$\det(V) = \prod_{0 \le l < t < k} (\hat{x}_t - \hat{x}_l),$$

given $\underline{\hat{x}} = (\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{k-1})^T = (x_{i,0}, x_{j,1}, \ldots, x_{l,k-1})^T$ the 2nd column of the Vandermonde matrix,

Hence, the determinant is non-null if and only if all the $\hat{x}_i$ -s are non-null and different from each other.

It should finally be noted that, to allow decoding Reed-Solomon codes, the encoding matrix has to be *known* at both the source and destination sites.

3.1.4. Arithmetic of RS-codes

Reed-Solomon codes are based on a mathematics area known as *Galois fields* or *finite fields*. A Galois field $GF(p)$, also called *prime field*, is a field that includes p elements, from *0* to *p-1*, with p prime. It is *closed* under additions and multiplications modulo p. Operating on a prime field is relatively simple, since field elements can be thought of as integers modulo p, and sums and products are just ordinary sums and products, with results computed modulo p. Galois Fields can be used for the representations of the elements involved in RS-codes, i.e., the blocks of source data, the blocks of encoded data, and the elements of the encoding matrix. However, from the implementation standpoint, this is not feasible or at least efficient for two main reasons. Firstly the number of bits necessary to represent an element of a Galois Field is $\lceil \log_2 p \rceil > \log_2 p$. This introduces inefficiency for encoding and also for computation of operations because the operand sizes

may not match the word size of the processor. In addition, operations modulo *p* are expensive because they need a division to be applied.

It is much more efficient to work on a so-called *Extension Field* $GF(p^r)$ that includes $q = p^r$ elements, with *p* prime and $r > 1$. Field elements can be represented as polynomials of degree $r-1$ with coefficients in $GF(p)$. Operations in Extension Fields with $p = 2$ can be extremely fast and simple to implement. Modulo operations are not needed. Efficient implementations can be realized exploiting only XORs and table lookups. A thorough dissertation on the arithmetic of Extension Field can be found, for example, in [31] and [32].

3.1.5. Systematic Codes

When the code-words include a verbatim copy of the source words, the code is said to be *systematic*. This corresponds to including the identity matrix I_k in the encoding matrix. The advantage of a systematic code is that it simplifies reconstruction of the source words in case very few losses are expected. In Fig. 3 for example, two out of the *k* code-words arrived at destination are actual original words. Hence, the system of equations that must be solved to reconstruct the original words includes *k-2* equations instead of *k*.

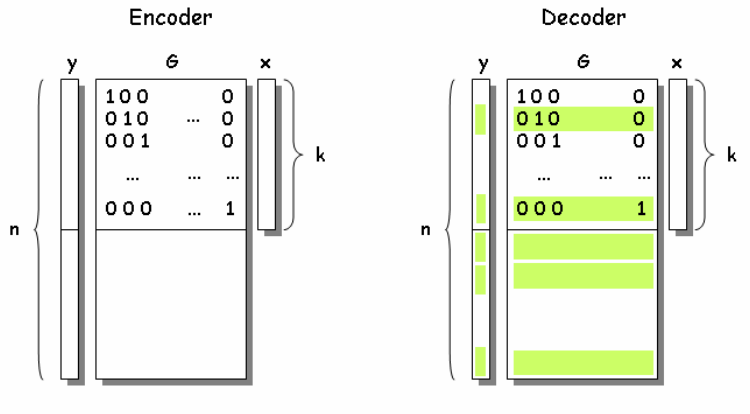

Fig. 3. The encoding/decoding process in matrix form, for systematic code (the top *k* rows of *G* constitute the identity matrix I_k). *y'* and *G'* correspond to the green areas of the vector and matrix on the right.

By using $x_i \in GF(q = p^r)$, it is possible to construct encoding matrices with up to *q-1* rows corresponding to the total number of non-null x_i's of the field. The number of columns of the encoding matrix is instead *k* corresponding, as usual, to the total number of original source words and the relation $q - 1 \geq k$ must hold true. As a result, the maximum size of the encoding matrix is $(q-1)\times(q-1)$.

The encoding matrix can further be extended with the identity matrix I_k to generate systematic codes that help simplify the decoding process. Hence, the final maximum size of the encoding matrix is as follows.

$$((q-1)+k_{\max})\times k_{\max} = ((q-1)+(q-1))\times(q-1) = 2(q-1)\times(q-1),$$

where $k_{\max}$ is the maximum number of source words that can be encoded over $GF(q)$.

3.1.6. Applications

A major hurdle in implementing a digital fountain through standard Reed-Solomon codes (RS codes) is the unacceptably high computational time caused by both encoding and decoding. It is of order $O(n^2)$, *n* corresponding to the number of generated code-words. The large decoding time for RS codes arises from the *dense* system of linear equations which is generated. However, efficient implementations of RS codes are actually feasible as has been shown in [22] and [23]. By means of thorough optimizations the developed code performs encoding and decoding on common microprocessors at speeds up to several MB/s. The code has been tested on a wide range of systems, from high end workstations to old machines and small portable systems (see e.g., DECstation 2100, SUN IPX, Sun Ultra1, PC/FreeBSD) with CPU speeds ranging from 8 up to 255 MHz. This variety is motivated by the fact that erasure codes can be used in very different contexts

and with very different speed requirements. Results effectively demonstrate the feasibility of *software FEC (Forward Error Correction)* and advocate its use in practical, real applications. The same code has finally been utilized to implement a reliable multicast data distribution protocol [8]. Namely, the use case referred is a large file transfer from a server to multiple receivers. The file is already encoded (off-line) and stored at the server ready for transmission. Receivers can join the multicast session whenever they need, generally at different time instants. The server needs a counter of the maximum number of encoded packets that have to be sent out to each receiver. This maximum number is the sum of the number of packets containing original data, and the number of packets containing redundant data. Each time a new receiver joins the session the server simply re-initializes the counter to the maximum value. Then, it continues transmitting the file from exactly the same point it had arrived when the last receiver joined the session and in the same order already begun. The counter is decremented after sending a packet. Once arrived to the end of the file, the server loops through the file until all the requests are fulfilled, i.e., until the counter goes to zero meaning that all the receivers have received the appropriate number of packets. Since with RS-codes reconstruction of source data is possible after having successfully received any *k*-set of distinct encoded packets (*k* being the size in packets of the original data set), each receiver leaves the session when it has received enough packets to decode the entire file. When a new receiver joins the session, transmission of data does not need to start again from the beginning of the file. The new receiver receives the same packets the other receivers are receiving, but it will stop receiving later than the others.

Fig. 4 shows the scheme followed by the server to transmit the file content. The source file is sub-divided into *B* blocks of data (on the vertical axis of the figure). The size of each block is such that it can be sent over *k* packets, thus it can be said that each block includes *k* packets. Each block is encoded separately, so redundancy is added to each single block independently from the others. Hence, assuming a systematic code, each encoded block includes *k* original packets plus *n-k* redundant packets (on the horizontal axis of the figure). Finally, the server sends out packets by repeatedly looping through all the blocks. Namely, packets scheduled for transmission are picked from consecutive blocks, rather than sequentially from the same block. Such *interleaving* protects single blocks from the effects of a burst of losses since it is preferable that losses are spread among all the blocks rather than concentrated in a unique block. Encoding is applied to single blocks of data rather than to the entire file to reduce the computational burden of both encoding and decoding. As has been mentioned above, in fact, the encoding and decoding costs are quadratic in the size of the encoded block ($O(n^2)$). Therefore, given that $n > k$, if *k* were the size of the entire file, the computational cost would be very high, whereas, by encoding over smaller chunks of file at a time is much more efficient (even though encoding must be repeated more times, once for each block).

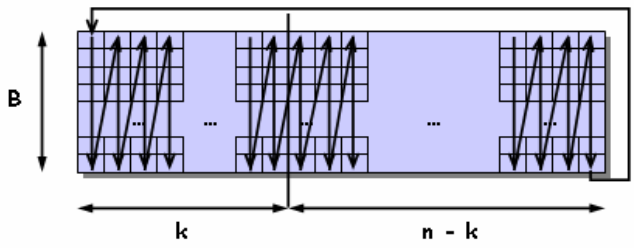

Fig. 4. The arrangement of data and the transmission order at the server.

The transmission scheme of Fig. 4 is known as *data carousel* (see also [24]). It limits both encoding and decoding times, allows multiple receivers to be served at the same time, and eliminates the need for several retransmissions towards different receivers. However, it has a drawback in the *receipt overhead*. In fact, since transmission loops through different blocks, after having completely received and decoded a certain number of blocks, a receiver can potentially receive and discard packets belonging to the blocks already decoded before receiving the missing packets necessary to decode the last blocks. The receipt overhead is measured in terms of unnecessary received packets (see Fig. 5).

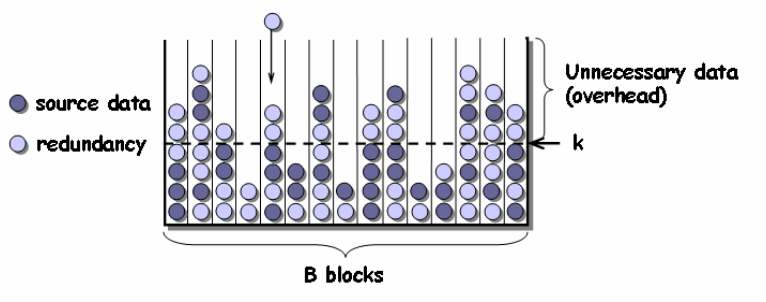

Fig. 5. Waiting for the last blocks to fill.

Starting from the achievements of [22], [23], and [8], an implementation of RS-Codes has been provided also for wireless sensor networks. Authors of [9] propose a point-to-point data transmission protocol for WSNs that *combines* multiple strategies, namely i) erasure codes, ii) re-transmission of lost packets and iii) a route-fix technique. RS codes are used to add redundancy to transmission such that the receiver is able to reconstruct original data despite losses or congestion (up to a certain point). Nevertheless, as underlined in [9], re-transmissions cannot be eliminated completely. Rather it is necessary to provide integration between encoding and re-transmission of data and to fine-tune this integration for better efficiency. Re-transmissions are managed hop-by-hop such that the point of retransmission is moved progressively forward towards the destination node. Experimental results show that these two strategies together are very efficient and result in good reliability. Nevertheless, it has also been noted that both link-level re-transmissions and erasure codes have poor performance in case of route failures. When a route is no more available, consecutive losses occur and re-transmissions are of no use because the route towards the destination does not exist any more. Neither erasure codes are helpful in this case because receivers cannot receive the minimum amount of encoded messages that is needed to reconstruct the original messages. A viable solution consists in finding an alternative route as soon as possible. Authors of [9] have also integrated route fixing into the framework with encoding and re-transmission of data and have used the Beacon Vector Routing (BVR) protocol [33] (a particular case of geographic routing for WSNs) at the routing layer to provide the special support needed. Results show improvements in both reliability of transmissions and RTTs experienced by messages.

3.2. Tornado Codes

Tornado Codes were introduced in 1997 to be fast, loss resilient codes, suitable to error-prone transmission channels. A preliminary version of these codes was presented in [26] and further improvements were later added in [27]. The overall structure of Tornado Codes is related to the low-density parity-check codes introduced by Gallagar in [34].

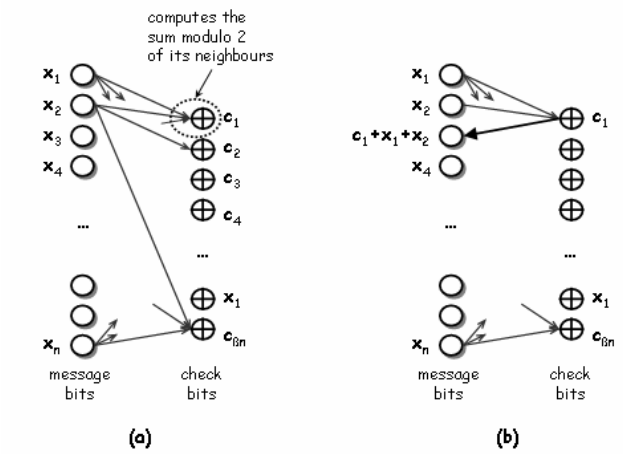

Fig. 6. (a) A bipartite graph defines a mapping from message bits to check bits. (b) Bits x_1, x_2, and c are used to solve for x_3.

Tornado codes cannot overtake the information-recovery capacity of RS codes, which is already optimal, but focus on faster encoding and decoding times. Namely, given n the length of the output block of the encoder corresponding to an input block of length k ($k < n$), encoding and decoding times of Tornado codes are of

order $O(n\ln(1/\varepsilon))$ for some $\varepsilon > 0$ (instead of $O(n^2)$ as in RS-codes). The price paid for much faster encoding and decoding times, with respect to RS codes, is that arrival of k packets is no longer sufficient for the receiver to reconstruct the source data.

3.2.1. Encoding Process

Suppose the source data to be encoded is a vector of k symbols in the finite field $GF(q)$ and let a symbol be a *bit*. The encoding process to yield the correspondent Tornado Code C relies on a bipartite graph B. Let the overall code be referred to as $C(B)$. It consists of a *systematic code* and thus includes the k bits of the original source message, named *message bits*, as well as $\beta \cdot k$ redundant bits ($0 < \beta < 1$) named *check bits*. The bipartite graph consists of k left nodes, $\beta \cdot k$ right nodes, and some edges to connect left nodes to right nodes. Left nodes correspond to message bits and right nodes correspond to check bits. Check bits are obtained by XOR-ing over the source message bits according to the bipartite graph B. Namely, a check bit is worked out by XOR-ing over all the message bits it is connected to in the graph B (see Fig. 6 (a)). The bipartite graph is sparse, random and irregular. Its construction is actually a bit complex, and is principally oriented to the reduction of decoding costs (see below).

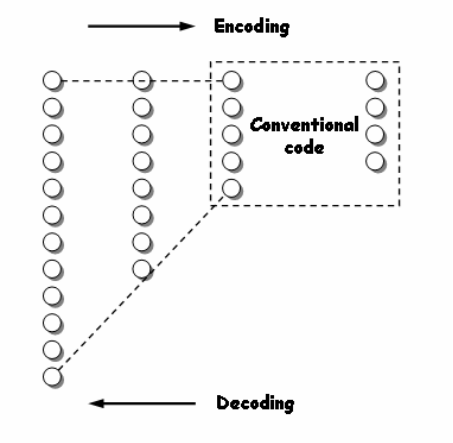

Fig. 7. The code levels.

By making things a bit more complex than has been described so far, a Tornado code is generally obtained with a multi-bipartite graph, i.e., a sequence (cascade) of successive bipartite graphs. In the first stage, a set of $\beta \cdot k$ redundant bits is generated from k message bits with a first bipartite graph, as explained above. Then, a second graph is added starting from the check bits of the previous graph. The new graph has thus $\beta \cdot k$ left bits and $\beta^2 \cdot k$ right bits. Then, a further graph can be added with $\beta^2 \cdot k$ left bits and $\beta^3 \cdot k$ right bits, and so on. More formally, it should be said that a set of codes $C(B_0), C(B_1), ..., C(B_m)$ is constructed by means of a set of bipartite graphs $B_0, B_1, ... B_m$ where a generic graph B_i has $\beta^i \cdot k$ left nodes and $\beta^{i+1} \cdot k$ right nodes. The value m is chosen such that $\beta^{m+1} \cdot k$ is roughly $\sqrt{k}$. The check bits of the last graph are finally encoded via another erasure code, say C, which can be, for example, a Reed-Solomon code as proposed in [27]. In Fig. 7 the last level erasure code is a systematic code with left nodes as original data and right nodes as redundant data. The final code includes the original k message bits, the check bits produced by the bipartite graphs at the m different levels, and the last-level RS-code. The entire code can be referred to as $C(B_0, B_1, B_2, ..., B_m, C)$. Given k the number of original message bits, the total number of check bits is as follows.

$$\sum_{i=1}^{m+1} \beta^i k + \beta^{m+2} k / (1-\beta) = k\beta(1-\beta).$$

Hence, the code $C(B_0, B_1, B_2, ..., B_m, C)$ has globally k message bits and $k\beta(1-\beta)$ check bits.

3.2.2. Decoding Process

Decoding relies on the assumption that the receiver knows the position of each received symbol within the graph and is performed going backward throughout the multi-bipartite graph. The RS-code C is decoded first. If the decoding process tackles to reconstruct all the original $\beta^{m+1} \cdot k$ bits of the RS-code then the last level of check bits is complete whereas the last but one level as well as all the previous ones still need reconstruction. The decoding process proceeds one level at a time and requires finding each time a check bit on the right side such that only one adjacent message bit (among those that have generated it) is missing. The missing message bit can thus be obtained by XOR-ing among the check bit and its known message bits. As an example (see Fig. 6-(b)), suppose c is a check bit whereas x_1, x_2, and x_3 the message bits it depends on i.e., $c = x_1 \oplus x_2 \oplus x_3$. Suppose also that, after transmission, c, x_1, and x_2 are known whereas x_3 is unknown. Then, the following holds true.

$$x_1 \oplus x_2 \oplus c = (x_1 \oplus x_2) \oplus (x_1 \oplus x_2) \oplus x_3 = x_3 .$$

The missing message bits are yielded one at a time by exploiting the graph relations in which they are involved together with other known message bits, to produce other known check bits. Once decoded a single level, the previous one is decoded then. By repeating the same procedure for each bipartite graph, the original k-bit message can finally be reconstructed, as well.

3.2.3. Properties of Tornado Codes

This sub-section provides some insights into interesting features of Tornado codes related to constraints and requirements of the encoding process, and some advantages and drawbacks too.

Property 1

The first property is about the feasibility of Tornado codes. This is strictly connected to the particular characteristics of the multi-bipartite graph in that successful decoding is only possible if an appropriate *degree distribution* is fulfilled during construction of the graph. The degree of a node belonging to the *k*-th graph level gives the number of nodes of the *(k-1)*-th level that node is generated by. [27] makes use of linear programming techniques to select the degrees to assign to the nodes of the same level and specifically defines appropriate *sequences* of node degrees.

Property 2

The shape of the multi-bipartite graph also impacts the performance of Tornado codes. Both encoding and decoding times strictly depend on the number of edges which are present in the multi-bipartite graph. Clearly, the higher is the number of edges, the higher is the computational burden of the code. The aforementioned encoding approach based on linear programming converges in *linear* time and also allows linear time decoding, as has been formally demonstrated in [27]. Limiting encoding and decoding complexity to linear time is also the reason why the last bipartite graph in the cascading sequence of bipartite graphs should have $\sqrt{k}$ right nodes (k being the number of nodes associated with the original message) and the total number of graphs in the sequence should consequently be $O(\log(k))$. $\sqrt{k}$ nodes are the ingress nodes of the last stage of encoding and, given the quadratic complexity of the RS-code which is used, by reducing the input size of the encoder, also the complexity is limited. The overall time complexity is about linear.

Property 3

Code efficiency of Tornado codes is worse than code efficiency of RS-codes. This is because Tornado codes need slightly more than k encoded bits to arrive to destination for successful decoding of the original k message bits. Namely, $1.063 \cdot k$ encoded bits, at least, must be collected at destination. However, better results have been obtained with an improved version of Tornado codes. This new version uses a three-level graph and does not perform RS-coding at the last level of the graph but rather repeats the same type of random bipartite graph as the previous two levels. Results show that $1.033 \cdot k$ encoded bits are needed with this version of Tornado codes to reconstruct the original k message bits.

Property 4

Some of the limits of Tornado codes are related to the *stretch factor* which is defined as the ratio n/k. Tornado codes can admit only a small stretch factor so as to limit the computational burden. A typical value is for example 4. However, a limited stretch factor limits in practice the applicability of Tornado codes. Another drawback related to the stretch factor is that, both when using RS code or another kind of erasure code at the last stage, the stretch factor must be *predetermined* before encoding takes place. Hence, researchers need to estimate, in advance, the loss probability of the channel to determine the amount of redundancy to put into the data. For most Internet applications, the loss probability varies widely over time. Thus, in practice, the loss rate has to be overestimated, making the codes slower to decode and far less efficient in transmitting information. If the loss probability is underestimated instead, the code fails.

3.3. LT-codes

With Luby-Transform codes (LT codes for short) it is no more necessary to tune the encoding process over the loss characteristics of the transmission channel. LT codes do not rely on a fixed rate to add redundancy to the data to send out, rather they are said to be *rateless* codes. They were first devised in 1998 and can definitely be considered the first full realization of the concept of digital fountains [28].

Let a symbol be an *l*-bit string. Encoding is performed over *k* input symbols and has the potential to generate a limitless number of encoded symbols. Each encoded symbol is produced, on average, by $O(\ln(k/\delta))$ symbol operations, being $0 < \delta \leq 1$. The original *k* input symbols can be reconstructed with probability $(1-\delta)$ from any set of $k + O(\sqrt{k}\ln^2(k/\delta))$ encoded symbols out of the total number of encoded symbols produced. The total number of operations needed to recover the original *k* symbols is about $O(k\ln(k/\delta))$. Hence, encoding and decoding times depend on the ingress data length independent of how many encoded symbols are generated and sent by the decoder. The same relations hold for the encoder and decoder memory usage. Therefore, given the reduced overall costs of LT codes with respect to the other aforementioned erasure codes, encoding can be performed over bigger chunks of data than those used in RS-codes and Tornado Codes, or even over the entire file at once.

Luby-Transform codes are similar in construction and properties to Tornado codes because they also rely on a graph and the encoded symbols are obtained by XOR-ing over other initial (or previously encoded) symbols (see Fig. 8). However, the graph construction for LT codes is quite different from Tornado codes and the resulting graph has logarithmic density. Moreover, LT codes are not systematic codes.

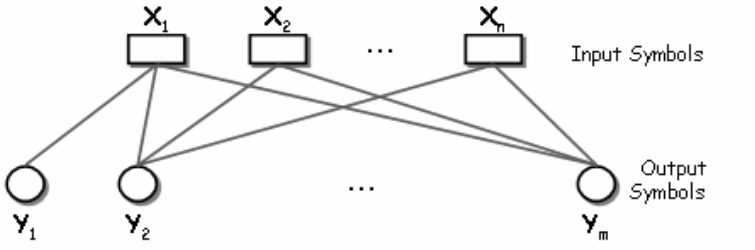

Fig. 8. Bipartite graph illustrating input symbols and output symbols.

3.3.1. Encoding Process

Let the input file to be encoded have length N. The file is partitioned into $k = N/l$ input symbols i.e., each input symbol has length l. Let an input symbol be referred to as a packet. Similarly, each output symbol is referred to as an encoded packet. The encoding operation defines a bipartite graph connecting output symbols to input symbols. For each output symbol (encoded packet) encoding is performed by **i)** randomly choosing the degree *d* from a given degree distribution, **ii)** choosing, uniformly at random again, *d* distinct input symbols as neighbours[4] of the encoded symbol, and **iii**) XOR-ing over the selected *d* input symbols taken as neighbours. The result of the bit-by-bit XOR among the selected symbols is the value of the encoded symbol. The degree distribution is computed easily independently of the data length (see below). To

[4] The neighbours of an encoded symbol are those symbols the previous layer of the bipartite graph to which the encoded symbol is connected by edges. An encoded symbol is obtained by XOR-ing over all its neighbours of the previous graph-layer.

work properly, the decoder should be able to reconstruct the bipartite graph without errors, i.e., it needs to know which *d* input symbols have been used to generate any given output symbol, but not their values.

3.3.2. Decoding Process

The decoder needs to know the degree and set of neighbours of each encoded symbol. Hence, for example, the degree and list of neighbour indices may be given explicitly to the decoder for each encoded symbol, or, as another example, a key may be associated with each encoded symbol and then both the encoder and the decoder apply the same function to the key to produce the degree and set of neighbours of the encoded symbol. The encoder may randomly choose each key it uses to generate an encoded symbol and keys may be sent to the decoder along with the encoded symbols.

The decoder receives K output symbols and the bipartite graph, from which it tries to recover the input symbols. Typically, K is slightly larger than k. The decoding process can be described as follows (see the Belief Propagation algorithm [35]).

1. Find an output symbol y_i that is connected to only one input symbol x_j. If there is no such output symbol, the decoding algorithm halts at this point, and fails to recover all the input symbols.
2. Set $x_j = y_i$.
3. Add (i.e., perform a bit-by-bit XOR) x_j to all the output symbols that are connected to x_j.
4. Remove all the edges connected to the input symbol x_j.
5. Repeat until all $\{x_j\}$ are determined or otherwise no more output symbols can be found that have exactly one neighbour. In the first case decoding ends successfully, in the latter, it fails.

LT codes require a simple decoder, but it turns out that the *degree distribution* is a critical part of the design. The decoding process as described above will not even start if there is no output symbol of degree 1. This means that a good degree distribution is required for good decoding performances. Indeed, whether or not the decoding algorithm is successful depends solely on the degree distribution. The degree distribution used to produce LT codes is the *Soliton Distribution* or the more efficient *Robust Soliton Distribution*. Further details on the degree distribution analysis and the Soliton and Robust soliton distributions can be found in [28].

3.3.3. Applications

In 1998 a new company (Digital Fountain Inc) was launched by the inventor of LT codes (Michael Luby) [36]. The company aimed at exploiting the digital fountain's technology as a tool for downloading popular software packages, such as Adobe Acrobat, and indeed, Adobe was an early investor, as were Cisco Systems and Sony Corporation. Digital Fountain Inc. took obviously a patent on LT codes and this has been limiting utilization of LT codes in the research community as well as in other research and development centres. Digital Fountain's researchers have been working on a new generation of codes and the group is also addressing other related research issues, such as application of the technology to video-on-demand and streaming media. Recently, the group has been focusing on congestion controls to ensure that the Digital Fountain packets behave in ways that don't cause network flow problems. Indeed, security issues are being addressed in collaboration with some universities to conceive, for example, a packet authentication system for verifying that data has come from a particular source. Launched to address the problems of streaming media and multicast, Digital Fountain -like many new technology companies- has found an unexpected niche. Without the requirement for TCP acknowledgments, data encoded with Luby transform codes travels faster than ordinary packets; this advantage becomes significant over long distances. Thus, the technology has proved particularly useful to companies that frequently transmit extremely large data files over long distances, such as movie studios and oil exploration companies.

Inspired to LT codes, the work in [11] proposes using digital fountains in wireless sensor networks. The sensor network is viewed as a big DataBase with k independent data-generating nodes and n data-storage nodes. Each storage node is assumed to have a limited storage capacity of only one single data packet. Data packets are small and generated independently from each other. It is indeed assumed that no correlation exists between them. The k source data packets are encoded and distributed after addition of redundancy to the n storage nodes. A collector/sink node wishing to retrieve information from the sensor network needs only to query any set of storage nodes of size slightly larger than k to reconstruct the entire data sensed at the source nodes. The solution proposed is inspired to the family of protocols named Sensor Protocols for Information via Negotiation (SPIN). These protocols assume that all the nodes have enough storage space to

store all the information gathered in the network. Hence the information gathered by the sensors is disseminated throughout the entire network and a user can query any node to get the required information immediately by only communicating with one node. The solution in [11] is different in that each sensor node has not enough memory to store all the data generated in the network and thus collects a combination of the total information gathered in the network. One single interrogation is not therefore sufficient but slightly more than *k* are required.

3.4. Raptor Codes

Raptor codes are a recent extension of LT codes (2004) being conceived with the aim to improve the decoding probability of LT codes [29]. The decoding graph of LT codes, in fact, needs to be of the order of $k\log(k)$ in the number of edges (k being the total number of input symbols) to make sure that all the input symbols can be recovered with high probability. Raptor codes focus on relaxing this condition such that the decoding process requires a smaller number of edges to be available. Raptor codes introduce a first level of encoding where the input symbols are pre-coded and a new set of symbols comprising some redundancy is generated. The new set of symbols is then given as input to a second level of encoding that performs LT coding (see Fig. 9).

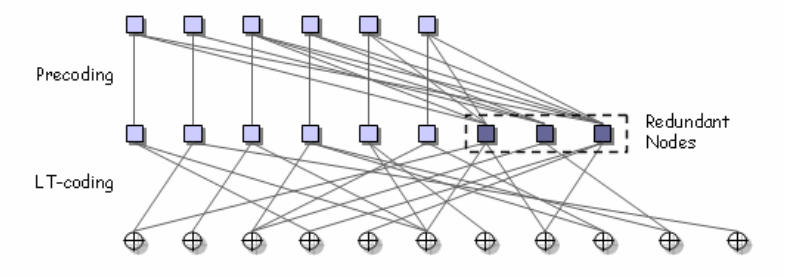

Fig. 9. Raptor Codes: the input symbols are appended by redundant symbols (dark squares) in the case of a systematic pre-code. An appropriate LT-code is used to generate output symbols from the pre-coded input symbols.

Pre-coding consists of a traditional erasure correcting code, say C, with a fixed stretch factor. The choice of code C depends on the specific application in mind. One possible choice is to use a Tornado code, but other choices are also possible such as, an LT-code with the appropriate number of output symbols. The second-level LT code needs appropriate tuning such that it is capable to recover all the input symbols (i.e., symbols given as input to pre-coding) even in face of a fixed fraction of erasures.

3.4.1. Encoding Process

Let C be a linear code producing n output symbols from k input symbols, and let D be a degree distribution. A *Raptor Code* is thus an LT code with distribution D on n symbols which are the coordinates of the code-words in C. Code C is called *pre-code* of the Raptor Code. The input symbols of a Raptor Code are thus the k symbols used to construct the code-word in C. The code-word consists of n *intermediate symbols*. Then, the output symbols are the symbols generated by the LT Code from the n intermediate symbols using the degree distribution D. Typically, C is a *systematic* code, though this is not strictly necessary. The encoding cost of a Raptor Code takes into account the encoding cost of both pre-coding and LT coding. It thus includes the number of arithmetic operations sufficient for generating a code-word in C from the k input symbols. The encoding cost is generally a per-symbol cost. Hence, it is worked out dividing the total cost to generate n output symbols by k.

3.4.2. Decoding Process

A reliable decoding algorithm of length m for a Raptor code is an algorithm which can recover the initial k input symbols from any set of m output symbols and errs with probability at most $1/k^c$ for some constant $c>1$. The decoding process will include in the first stage an LT decoding process that will recover a fraction of the n intermediate symbols that have been given as input to the LT encoder in the encoding phase. Then, the second stage will consist of a classical erasure decoding process that should be able to recover the k original input symbols of the entire Raptor encoder. The decoding cost of a Raptor Code is the

expected number of arithmetic operations sufficient to recover the k input symbols, divided by k. However, the overall efficiency of Raptor codes should not be measured only in terms of the number of operations per single input symbol but also in terms of *space* and *overhead*. Space refers to the memory consumption for storage of intermediate symbols. Overhead instead is a function of the decoding algorithm used, and is defined as the number of output symbols that the decoder needs to collect in order to recover the input symbols with high probability.

Different types of Raptor Codes can be obtained depending on the type of erasure code C which is used for pre-coding and the particular degree distribution used in the second level encoding. Hence, a complex pre-coding[5] can be combined with a simple degree distribution of the LT encoder or vice-versa a simple pre-coding algorithm can be combined with a complex degree distribution. In between these two extremes other solutions are also possible.

Optimized Raptor Codes applied to k original input symbols have the potential to generate an infinite stream of output symbols. Any subset of symbols of size $k(1+\varepsilon)$, for some $\varepsilon > 0$, is then sufficient to recover the original k input symbols with high probability. The number of operations needed to produce each output symbol is of order $O(\log(1/\varepsilon))$ whereas order $O(k \cdot \log(1/\varepsilon))$ operations are needed to recover all the k original data symbols. Differently from LT codes, Raptor codes can be *systematic codes*.

3.4.3. Applications

Raptor Codes are being used in commercial systems by Digital Fountain Inc. for fast and reliable delivery of data over heterogeneous networks. The Raptor implementation of Digital Fountain reaches speeds of several gigabits per second, on a 2.4 GHz Intel Xeon processor, while ensuring very stringent conditions on the error probability of the decoder.

[5] Sometimes complex erasure codes can be used when intermediate symbols (code-words in C) can be calculated off-line via pre-processing.

4. Network Coding

Network Coding is recently emerging as a new promising research field in the networking framework [37]. Although sharing some basic concepts with erasure coding and digital fountains, network coding differs from the aforementioned encoding techniques in many interesting aspects. First of all, the target of network coding is something different from that of digital fountains. Network coding has been conceived to allow *better resource utilization* during transmission over the network, especially as regards *bandwidth* and *throughput*. In addition, network coding allows *optimal delays* and better traffic distribution throughout the network thus leading to overall *load balancing* which is very promising in scenarios where power saving is a major concern (see e.g., wireless ad hoc and sensor networks). It adds *robustness* against node and connections failures (even permanent) as well as *flexibility* against changes in the network topology [38]. Network Coding therefore is not concerned with reliable transmissions, which is instead the major target of Erasure Coding, but focuses more generally on the overall optimization of network usage. Moreover, as far as robustness and flexibility are concerned, network coding seems to be quite suitable to new emerging scenarios of extreme and challenged networks like, for example, Delay Tolerant Networks (DTNs), intermittently connected ad hoc and sensor networks, underwater acoustic sensor networks, and vehicular ad hoc networks. These scenarios are prone to link failures because nodes can move, crash or simply go to sleep-mode to save energy. It has been shown that network coding-based algorithms for data dissemination where all nodes are interested in knowing all the information produced, are actually very efficient.

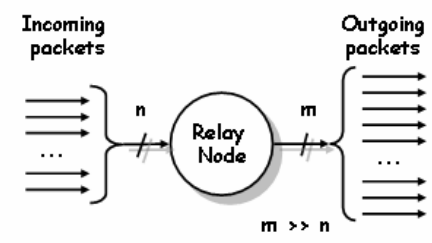

Fig. 10. Relay nodes perform encoding over *n* ingress data packets and produce *m* data packets to send out. The number of encoded packets is greater than the number of incoming data packets.

Apart from targets and performance, the other great difference between erasure and network coding is recursive coding at intermediate nodes. With network coding, both source nodes and intermediate/relay nodes encode the original and/or to-be-forwarded data while in generic relay systems relay nodes simply repeat data towards the next identified hops (see Fig. 10).

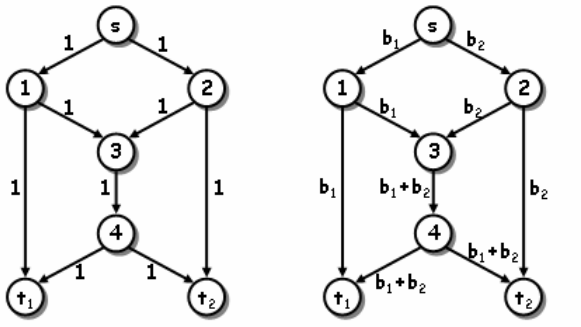

Fig. 11. A one-source two-sink network with coding: link capacities (left) and network coding scheme (right).

To illustrate the advantages of network coding let consider a classical example. Fig. 11 shows the graph representation of a one-source two-sink network. The capacity of each link (edge) of the network is one bit per second. The value of the max-flow from the source node to each of the destination nodes t_1 and t_2, according to the well known max-flow min-cut theorem is 2. So it should be possible to send 2 bits, b_1 and b_2, to t_1 and t_2 simultaneously.

However, this is only possible by using network coding. Fig. 11 shows the network coding scheme that allows the max-flow. "+" denotes XOR operation. Instead of forwarding b_1 and b_2 in two separate

transmissions, node 3 encodes them and sends just a single encoded packet, b_1+b_2, which eventually reaches t_1 and t_2 via node 4. As node t_1 and t_2 also get respectively b_1 and b_2 via different routes, they are able to recover the missing bit from the encoded packet received from node 3.

The above network scheme is well known and called "butterfly network". It was presented in 2000 in [37], when the term network coding was originally proposed. The same article also demonstrated that network coding is necessary to achieve the theoretical upper bound limit for throughput utilization (max-flow) in optimal multicast schemes that include two or more destination nodes. However, throughput benefits of network coding are not only limited to multicast flows [39] and can extend to other traffic patterns, e.g., unicast transmissions [40]. Throughput gains are different depending on the network graph. Specifically, they can be very significant in directed graphs whereas, in undirected graphs (e.g., a wired network where all links are half-duplex) the throughput gain is at most a factor of two [41] [42].

In the following sections 4.1 and 4.2, some insights on the encoding and decoding processes of network coding will be given. Indeed, a particular type of network coding will be described which is called *random, linear network coding* [43]. Linear coding is one of the simplest coding schemes for network coding. It regards a block of data as a vector over a certain base field and allows a node to apply a linear transformation to this vector before forwarding it [44]. When the transformation is performed at each node independently from other nodes, and using random coefficients, linear coding is said to be random network coding. Its efficacy has been extensively studied, mainly analytically, for different types of networks corresponding to different graphs [45], [46], and [47]. Cases that have been studied include: i) one-source networks and networks with multiple sources either independent or linearly correlated; ii) acyclic networks and cyclic networks; iii) delay-free and delayed networks. In all these environments random network coding has brought interesting and valuable benefits.

Finally, some recent applications of Network Coding will be described in Section 4.3.

4.1. Encoding Process

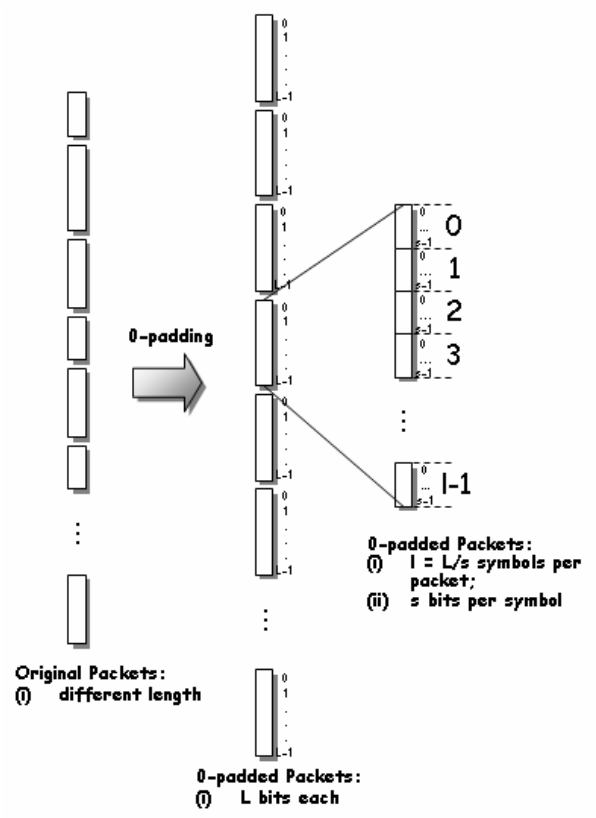

Fig. 12. All the packets have the same length of L bits. Shorter packets are 0-padded.

As mentioned before, encoding is performed both at the source and intermediate relay nodes which are visited by the data flowing over the network. Data is organized into packets. Assume that all the incoming/generated *packets* have the same length of *L bits.* If a packet is shorter, then it is padded with trailing 0s. The content of a packet can be interpreted as a sequence of *symbols*, where each symbol corresponds to a sequence of *s* consecutive bits. Hence, each 0-padded packet, L-bit long, is a sequence of *L/s*, say *l*, symbols. Each symbol belongs to the field $GF(2^s)$ since it can be represented over *s* bits.

In Linear Network Coding the encoding process works out a linear combination over a set of incoming packets (at a relay node) or original packets (at the source) and produces new packets for sending out, which have the same size (*L* bits) of the packets that have been combined together. The arithmetic operations involved in linear combinations (addition and multiplication) are performed over the field $GF(2^s)$.

Assume having a set of *n* ingress packets, namely $M^0, M^1, ..., M^{n-1}$. Each packet is a vector of $l = L/s$ symbols. Ingress packets are *original packets* in case of source nodes, whereas, in case of relay nodes, they may be original and *incoming packets* as well. Incoming packets are intended to come from one or even multiple nodes. However, suppose for simplicity that the incoming packets come all from the same source node.

The node that performs encoding starts generating *n* coefficients, one for each ingress packet: $g_0, g_1, ..., g_{n-1}$. Coefficients are symbols, i.e., $g_i \in GF(2^s)$. Then the node generates an outgoing packet X of size *L* bits as follows.

$$X = \sum_{i=0}^{n-1} g_i M^i .$$

With much detail the encoding process for a single outgoing packet *X* works as follows.

$$X = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ .. \\ .. \\ x_{l-1} \end{pmatrix} = \sum_{i=0}^{n-1} g_i M^i = \sum_{i=0}^{n-1} g_i \cdot \begin{pmatrix} m_0^i \\ m_1^i \\ m_2^i \\ .. \\ .. \\ m_{l-1}^i \end{pmatrix} = g_0 \cdot \begin{pmatrix} m_0^0 \\ m_1^0 \\ m_2^0 \\ .. \\ .. \\ m_{l-1}^0 \end{pmatrix} + g_1 \cdot \begin{pmatrix} m_0^1 \\ m_1^1 \\ m_2^1 \\ .. \\ .. \\ m_{l-1}^1 \end{pmatrix} + g_2 \cdot \begin{pmatrix} m_0^2 \\ m_1^2 \\ m_2^2 \\ .. \\ .. \\ m_{l-1}^2 \end{pmatrix} + ... + g_{n-1} \cdot \begin{pmatrix} m_0^{n-1} \\ m_1^{n-1} \\ m_2^{n-1} \\ .. \\ .. \\ m_{l-1}^{n-1} \end{pmatrix},$$

where $x_i, m_i^j, g_j \in GF(2^s)$. x_i is the *i-th* symbol of the egress vector (packet) *X*. m_i^j is the *i-th* symbol of the *j-th* ingress packet M^j. g_j is the coefficient that multiplies the *j-th* ingress packet M^j. The same encoding process is then repeated more times. Specifically, from *n* ingress packets a node generates *m* outgoing, egress packets with $m >> n$ so as at the receiving site decoding is possible after having received *any n out of m* packets. Hence, the encoding node actually generates *m* sets of coefficients, one for each egress packet to generate, and then performs encoding as follows (all the three formulas below are equivalent).

$$X^j = \sum_{i=0}^{n-1} g_i^j M^i; \quad j = 0..m-1,$$

$$\begin{pmatrix} X^0 \\ X^1 \\ X^2 \\ .. \\ .. \\ .. \\ X^{m-1} \end{pmatrix} = \begin{pmatrix} g_0^0 & g_1^0 & g_2^0 & .. & g_{n-1}^0 \\ g_0^1 & g_1^1 & g_2^1 & .. & g_{n-1}^1 \\ g_0^2 & g_1^2 & g_2^2 & .. & g_{n-1}^2 \\ .. & .. & .. & .. & .. \\ & & & & \\ .. & .. & .. & .. & .. \\ g_0^{m-1} & g_1^{m-1} & g_2^{m-1} & .. & g_{n-1}^{m-1} \end{pmatrix} \cdot \begin{pmatrix} M^0 \\ M^1 \\ .. \\ .. \\ M^{n-1} \end{pmatrix},$$

$$X_{(m\,x\,1)} = G_{(m\,x\,n)} \cdot M_{(n\,x\,1)},$$

where X^i is the *i-th* egress packet that is generated and g_i^j is the *i-th* symbol of the *j-th* encoding vector, i.e., the *j-th* set of coefficients which is generated to produce the *j-th* encoded packet. All the cofficients g_i^j together form the (mxn)-sized encoding matrix G. Each encoded data packet X^j is named *information vector* and the corresponding vector of coefficients it is obtained from, g^j, is said to be an *encoding vector*. The encoding node produces and sends out packets by including both encoding and information vectors, thus an outgoing packet is actually a tuple $< g^j, X^j >$ for $j = 0..m-1$. Each encoding vector includes *n* symbols, whereas each information vector includes *l* symbols, as many as in the ingress data items $M^i, \quad i = 0..n-1$.

$$< g^j, X^j > = < (g_0^j, g_1^j, g_2^j, ..., g_{n-1}^j), (x_0^j, x_1^j, x_2^j, ..., x_{l-1}^j) > .$$

When an encoding vector includes all zeros but one single one in the *i-th* position, it means that the *i-th* ingress data item is not encoded.

$$g^i = e^i = (0, 0, ..., 0, 1, 0, ..., 0) \quad \Rightarrow \quad X^i = M^i .$$

4.1.1. Recursive encoding

Suppose that the following conditions hold.

(i) An intermediate relay receives and stores a set of *r* tuples, as follows.

$$< g^0, X^0 >, < g^1, X^1 >, < g^2, X^2 >, ..., < g^{r-1}, X^{r-1} > .$$

(ii) All the information vectors received ($X^0, X^1, X^2, ..., X^{r-1}$) come from the same set of original data ($M^0, M^1, M^2, ..., M^{n-1}$) produced by a single source node somewhere.

Then, the intermediate node performs re-encoding on the set of information vectors that it has received: $X^0, X^1, X^2, ..., X^{r-1}$. It is not necessary that $r >> n$ because the intermediate node does not need decoding at this stage. It simply re-encodes the pieces of data it has received. This results in adding some more redundancy into the network for some part of the original data. Re-encoding is performed by generating a certain number of encoding vectors, as many as the number of information vectors that the node wants to send out. Suppose, for example, that the node wants to generate *k* information vectors. It should be noted that there is not a mathematical constraint on the values allowed for *r* and *k*. Rather, the right tuning of these parameters depends on the networking scenario and is still a challenging open issue.

After having chosen the appropriate value for *k*, the relay node acts as follows.

(a) Selects *k* new encoding vectors h^j s, $j = 0..k-1$. Each encoding vector includes as many symbols as the number of ingress information vectors the new encoding is applied to. In this case each encoding vector is *r*-symbol long.

$$h^j = (h_0^j, h_1^j, h_2^j, ..., h_{r-1}^j); \quad j : 0, .., k-1 .$$

(b) Produces *k* new information vectors as follows.

$$Y^j = \sum_{i=0}^{r-1} h_i^j X^i; \quad j = 0..k-1.$$

The new encoding matrix is therefore as follows.

$$H_{(k\ x\ r)} = \begin{pmatrix} h_0^0 & h_1^0 & .. & h_{r-1}^0 \\ h_0^1 & h_1^1 & .. & h_{r-1}^1 \\ .. & .. & .. & .. \\ h_0^{k-2} & h_1^{k-2} & .. & h_{r-1}^{k-2} \\ h_0^{k-1} & h_1^{k-1} & .. & h_{r-1}^{k-1} \end{pmatrix}.$$

Whereas the overall re-encoding process can be represented as follows.

$$\begin{pmatrix} Y^0 \\ Y^1 \\ .. \\ Y^{k-2} \\ Y^{k-1} \end{pmatrix} = \begin{pmatrix} h_0^0 & h_1^0 & .. & h_{r-1}^0 \\ h_0^1 & h_1^1 & .. & h_{r-1}^1 \\ .. & .. & .. & .. \\ h_0^{k-2} & h_1^{k-2} & .. & h_{r-1}^{k-2} \\ h_0^{k-1} & h_1^{k-1} & .. & h_{r-1}^{k-1} \end{pmatrix} \cdot \begin{pmatrix} X^0 \\ X^1 \\ .. \\ X^{r-1} \end{pmatrix} =$$

$$= \begin{pmatrix} h_0^0 & h_1^0 & .. & h_{r-1}^0 \\ h_0^1 & h_1^1 & .. & h_{r-1}^1 \\ .. & .. & .. & .. \\ h_0^{k-2} & h_1^{k-2} & .. & h_{r-1}^{k-2} \\ h_0^{k-1} & h_1^{k-1} & .. & h_{r-1}^{k-1} \end{pmatrix} \cdot \begin{pmatrix} g_0^0 & g_1^0 & g_2^0 & .. & g_{n-1}^0 \\ g_0^1 & g_1^1 & g_2^1 & .. & g_{n-1}^1 \\ .. & .. & .. & .. & .. \\ g_0^{r-1} & g_1^{r-1} & g_2^{r-1} & .. & g_{n-1}^{r-1} \end{pmatrix} \cdot \begin{pmatrix} M^0 \\ M^1 \\ M^2 \\ .. \\ M^{n-1} \end{pmatrix} \stackrel{\Delta}{=} G'_{(kxn)} \cdot \begin{pmatrix} M^0 \\ M^1 \\ M^2 \\ .. \\ M^{n-1} \end{pmatrix},$$

where $G'_{(kxn)}$ is equal to $H_{(kxr)} \cdot G_{(rxn)}$ and is hereafter called re-encoding matrix.

(c) Produces the *k* final encoding vectors to include in the outgoing tuples. They correspond to the rows of G' worked out as follows.

$$g_i'^j = \sum_{t=0}^{r-1} h_t^j \cdot g_i^t; \quad i:0..n-1; \quad j:0..k-1; \quad t:0..r-1.$$

(d) Sends out *k* tuples as follows.

$$< g'^0, Y^0 >, < g'^1, Y^1 >, < g'^2, Y^2 >, ..., < g'^{k-1}, Y^{k-1} >.$$

This process is equivalent to *encoding only once* the original data $M^0, M^1, ..., M^{n-1}$ with the new set of encoding coefficients. At the receiver site the number of incoming tuples to collect prior to decoding corresponds to the number of symbols included in a single encoding vector or, from another standpoint, to the number of original data packets (as can be seen, it is always *n*). Finally, it should be noticed that the outgoing information vector is again *l*-symbol long, as follows.

$$Y^i = (y_0^i, y_1^i, y_2^i, ..., y_{l-1}^i)^T.$$

The computational cost of performing re-encoding at each intermediate node depends on the following operations.

(i) The product between the new encoding matrix, $H_{(k\,x\,r)}$, and the old encoding matrix, $G_{(r\,x\,n)}$, to obtain the final encoding matrix to be included in the outgoing tuples: $G'_{(k\,x\,n)} = H_{(k\,x\,r)} \cdot G_{(r\,x\,n)}$.

(ii) The product between the new encoding matrix, $H_{(k\,x\,r)}$, and the matrix of encoded data, $X_{(r\,x\,l)}$, to obtain the outgoing information vectors $Y_{(k\,x\,l)}$.

4.1.2. How to choose encoding vectors?

The choice of coefficients to include in encoding vectors is, as can easily be expected, quite critical. An inaccurate choice can produce tuples which are linearly dependent on each other. When the destination node receives a tuple which is linearly dependent on another tuple already received, it discards the new tuple because it does not contain innovative information and is thus of no use (see Section 4.2 on the Decoding Process for further details). This directly impacts the decoding time causing the destination node to wait for more packets than strictly necessary to be able to do the decoding. A simple algorithm to generate such coefficients is proposed in [48] and gives rise to the so-called *Random Linear Coding.* Each node in the network selects *uniformly at random* its coefficients over the field *GF(2^s)* in a completely *independent* and *decentralized* manner. Simulation results indicate that even for small field sizes (e.g., with s=8), the probability of selecting linearly dependent combinations is *negligible.*

Other approaches have been proposed in [45] and [49]. The first one refers to a centralized algorithm where a single entity decides which coefficients each node in the network has to assign. The latter describes deterministic decentralized algorithms that apply to restricted families of network configurations.

4.1.3. Generations

Generation is the name assigned to a set of vectors (packets) which are encoded together at a source node. Typically, in fact, a source node produces a continuous flow of packets but obviously it can not apply encoding to them all at the same time because otherwise i) it should wait for the entire packet flow to be generated to perform encoding thus causing long delay to transmission, and ii) it should generate far long encoding vectors to include in outgoing packets. This would cause severe overhead in transmission.

Therefore, source vectors are grouped into generations and each generation has a corresponding matrix (that includes all the encoding vectors). To allow successful decoding at destination, it is necessary that only vectors (packets) belonging to the same generation are combined. This implies that packets should carry some information about the generation they belong to or, at least, this should be deducible in some other way so as intermediate nodes do not encode together packets of different generations thus making them impossible to decode.

4.2. Decoding Process

In Network Coding decoding is only needed at destination. Suppose *n* source data packets have been generated in the network. Let them be $M^0, M^1, ..., M^{n-1}$, with $M^i = \left(m_0^i, m_1^i, m_2^i, ..., m_{l-1}^i\right)$. Suppose that the receiver node has just received *r* data packets containing *r* tuples:

$$< g^0, X^0 >, < g^1, X^1 >, < g^2, X^2 >, ..., < g^{r-1}, X^{r-1} > .$$

Summing up, the encoding vectors $g^0, g^1, ..., g^{r-1}$ are such that $g^i = \left(g_0^i, g_1^i, g_2^i, ..., g_{n-1}^i\right)$ whereas the *r* encoded data packets $X^0, X^1, ..., X^{r-1}$ are such that $X^i = \sum_{j=0}^{n-1} g_j^i \cdot M^j$, and the following relations hold true.

$$x_z^i = \sum_{j=0}^{n-1} g_j^i \cdot m_z^j; \quad \Rightarrow \quad \begin{pmatrix} x_0^i \\ x_1^i \\ x_2^i \\ .. \\ x_{l-1}^i \end{pmatrix} = g_0^i \cdot \begin{pmatrix} m_0^0 \\ m_1^0 \\ m_2^0 \\ .. \\ m_{l-1}^0 \end{pmatrix} + g_1^i \cdot \begin{pmatrix} m_0^1 \\ m_1^1 \\ m_2^1 \\ .. \\ m_{l-1}^1 \end{pmatrix} + g_2^i \cdot \begin{pmatrix} m_0^2 \\ m_1^2 \\ m_2^2 \\ .. \\ m_{l-1}^2 \end{pmatrix} + ... + g_0^{n-1} \cdot \begin{pmatrix} m_0^{n-1} \\ m_1^{n-1} \\ m_2^{n-1} \\ .. \\ m_{l-1}^{n-1} \end{pmatrix}.$$

The above relations can also be written as follows.

$$X_{(r\ x\ 1)} = G_{(r\ x\ n)} \cdot M_{(n\ x\ 1)} \quad \Leftrightarrow$$

$$\begin{pmatrix} X^0 \\ X^1 \\ X^2 \\ .. \\ .. \\ .. \\ X^{r-1} \end{pmatrix} = \begin{pmatrix} g_0^0 & g_1^0 & g_2^0 & .. & g_{n-1}^0 \\ g_0^1 & g_1^1 & g_2^1 & .. & g_{n-1}^1 \\ g_0^2 & g_1^2 & g_2^2 & .. & g_{n-1}^2 \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ g_0^{r-1} & g_1^{r-1} & g_2^{r-1} & .. & g_{n-1}^{r-1} \end{pmatrix} \cdot \begin{pmatrix} M^0 \\ M^1 \\ .. \\ .. \\ M^{n-1} \end{pmatrix}.$$

To go back to the original messages $M^0, M^1, ..., M^{n-1}$ it is necessary that the following conditions hold true.

(i) $r \geq n$;

(ii) The encoding matrix $G_{(r\ x\ n)}$ has rank *n*. This happens if and only if the encoding matrix includes *n* rows linearly independent.

Provided conditions (i) and (ii) are met, the original data $M^0, M^1, ..., M^{n-1}$ can be derived via standard linear system solution techniques. A computational-efficient way of doing this on-line can be sketched as follows. If *r* packets have been received, a decoding matrix can be constructed as:

$$\left(\begin{array}{ccccccc|cccc} g_0^0 & g_1^0 & g_2^0 & g_3^0 & .. & g_{n-1}^0 & x_0^0 & x_1^0 & .. & x_{l-1}^0 \\ g_0^1 & g_1^1 & g_2^1 & g_3^1 & .. & g_{n-1}^1 & x_0^1 & x_1^1 & .. & x_{l-1}^1 \\ g_0^2 & g_1^2 & g_2^2 & g_3^2 & .. & g_{n-1}^2 & x_0^2 & x_1^2 & .. & x_{l-1}^2 \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ & & & & & & & & & \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ g_0^{r-2} & g_1^{r-2} & g_2^{r-2} & g_3^{r-2} & .. & g_{n-1}^{r-2} & x_0^{r-2} & x_1^{r-2} & .. & x_{l-1}^{r-2} \\ g_0^{r-1} & g_1^{r-1} & g_2^{r-1} & g_3^{r-1} & .. & g_{n-1}^{r-1} & x_0^{r-1} & x_1^{r-1} & .. & x_{l-1}^{r-1} \end{array}\right).$$

By gaussian elimination it is possible to reduce the system to the following one in which the original messages can be read.

$$\left(\begin{array}{c|cccc} & m_0^0 & m_1^0 & .. & m_{l-1}^0 \\ & m_0^1 & m_1^1 & .. & m_{l-1}^1 \\ I_{(n\ x\ n)} & m_0^2 & m_1^2 & .. & m_{l-1}^2 \\ & .. & .. & .. & .. \\ & m_0^{n-1} & m_1^{n-1} & .. & m_{l-1}^{n-1} \\ \hline G'_{((r-n)\ x\ n)} & \multicolumn{4}{c}{X'_{((r-n)\ x\ l)}} \end{array}\right).$$

To construct this matrix dynamically, as packets arrive to the receiver, they are added to the decoding matrix row by row (a row is $<(g_0^i, g_1^i, g_2^i, ..., g_{n-1}^i), (x_0^i, x_1^i, ..., x_{l-1}^i)>$). The *i-th* information vector X^i is said to be *innovative* if it increases the rank of the matrix. If a packet is not innovative, it is reduced to a row of all 0s by Gaussian elimination and is ignored. While performing gaussian elimination, as soon as the matrix contains a row of the form $<e_i, X>$, the node knows that the original packet M^i is equal to X (remember that we have defined e_i in Section 4.1 as a vector that includes all zeros but one single one in the *i-th* position).

4.3. Applications

In [19] Network Coding has been proposed to allow *efficient communication in extreme networks* i.e., in delay-tolerant networks or, more generally, intermittently connected networks. According to the forwarding scheme described, nodes do not simply forward packets they overhear but send out information that is coded over the content of several packets they have received. Simulation results show that this algorithm achieves the reliability and robustness of flooding at a small fraction of the overhead. Two interesting topics have been discussed, namely generations and redundancy.

The solution for generation recognition which has been proposed in [19] is as follows. Let $X_{(i)}^j$ be the *j-th* information vector that originates at the *i-th* source node S_i. Then a function $f(X_{(i)}^j)$ determines which generation the packet belongs to and $\Gamma_\gamma = \left\{X_{(i)}^j \middle| f(X_{(i)}^j) = \gamma\right\}$ is the set of all the source vectors of a generation γ. Namely, the generation membership is determined through hashing over the *sender address* and the *packet identifier*. A new hash function is generated whenever the matrix becomes too big. The hash function is then used at a node to determine which generation to insert a given packet into, provided the size of this generation does not exceed a certain max threshold. It has been demonstrated that managing

generations through hashing works better than simply incrementing the generation index from time to time, especially in ad hoc networks.

Another interesting topic raised in [19] is on the forwarding strategy and specifically, on how to decide when to send a new packet. The solution that has been proposed relies on a so-called *forwarding factor* $d > 0$ and establishes that whenever an innovative packet is received or generated at a node for a given generation, this has to forward a certain number of packets depending on the value of d and on whether the node is a source or an intermediate node. Specifically, in case of an intermediate node, it first generates $\lfloor d \rfloor$ vectors from the corresponding matrix and re-broadcast them to the neighbours, then it generates and sends a further vector with probability $d - \lfloor d \rfloor$. In case of a source node, whenever it generates a new original packet it encodes and broadcasts to the neighbours $\max(1, \lfloor d \rfloor)$ vectors. It then produces and sends out a further vector if $d > 1$ with probability $d - \lfloor d \rfloor$. The delivery policy at source nodes is obviously a bit different since at least one packet must be generated from each newly produced packet. In other words, a new packet is sent out by the source at least once.

With the Network Coding model described above, a node sends out on average $dG + 1$ packets where G is the maximum generation size ($G = m$ in the simplest case). Since receivers can decode all original vectors when they receive a number of innovative packets equal to the generation size, a good value for *d* strongly depends on the number of neighbours (i.e., the node density). Similar to probabilistic routing, a high forwarding factor results in a high decoding probability at the expense of a high network load.

In [50] and [51] network coding is used in a peer-to-peer content distribution network named *Avalanche*. In a peer-to-peer content distribution network a server splits a large file into a number of blocks, then peer nodes try to retrieve the original file by downloading blocks from the server but also distributing downloaded blocks among them. When using network coding, the blocks sent out by the server are random linear combinations of all the original blocks. Similarly, peers send out random linear combinations of all the blocks available to them. Network Coding in *Avalanche* minimizes download times with respect to the case it is not used. Network Coding gives the system more robustness in case the server leaves early or when peers only join for short periods.

Finally, some promising application fields for Network Coding are Wireless Ad Hoc, Mesh and Sensor networks where it can contribute to achieving throughput gains and more secure systems.

Network coding seems to facilitate protection from eavesdroppers, since information is more spread out and thus more difficult to “overhear”. It also simplifies the protection against modified packets in a network. In a network with no additional protection, an intermediate attacker may make arbitrary modifications to a packet to achieve a certain reaction at the attacked destination. However, in the case of network coding, an attacker cannot control the outcome of the decoding process at the destination, without knowing all other coded packets the destination will receive. Given that packets are routed along many different paths, this makes controlled man-in-the-middle-attacks more difficult.

5. Conclusions and Open Issues

Encoding techniques are becoming popular in the evolving Internet scenario. The advent of wireless technologies, the diffusion of an ever greater and heterogeneous number of portable devices, the diffusion of multimedia and content-distribution applications as well as their adoption over wireless networks are posing and renewing a number of networking challenges, such as reliability, scalability and efficient data distribution over wireless networks.

Encoding techniques have successfully been used to increase reliability and scalability, for example in multicast scenarios, in parallel download from many sources, and over overlay networks. Moreover, they are commonly applied to multimedia streaming applications to manage reliability issues without requiring feedback channels, and minimizing synchronizations and interactions among multiple receivers, and between senders and receivers.

Finally, it should be noted that the advantages of encoding are not only limited to transmission reliability and scalability but also regard security. It is much harder, for example, for the attackers to catch useful information from a network because they should have to collect a sufficiently high number of data packets that need to be decoded together and they should also know how exactly to do the decoding.

The main drawback of the encoding techniques analysed in this chapter is the *computational burden* involved in both the encoding and decoding processes. Also the *delay* might be negatively impacted being increased by the time spent in encoding and decoding and the time spent to transmit the extra-information produced during encoding (encoded data has higher size than the original). However, many optimizations have been studied and implementations of encoding techniques have also been provided even for Wireless Sensor Networks. Since WSNs are particularly scarce in resources, this means that resource usage has successfully been minimized.

Network Coding is a sort of generalization of encoding techniques. Most of the work conducted on network coding is analytical, so far, since it has been introduced as a way to improve use of network bandwidth. It has been shown that network coding allows achievement of the *max* possible *flow* on the networks under investigation including many different topology patterns. Network coding has also the advantage to increase *robustness* of the network against node failures, even permanent, and *adaptability* to variations of the network topology. By disseminating information throughout the network, network coding also provides *load balancing* of the network traffic, and, in addition, it adds *reliability* to transmissions as the other encoding techniques do.

The main drawback of network coding is the *computational burden* which is added not only to the source and destination nodes but also to the intermediate nodes that act as relays and provide re-encoding together with usual forwarding. Another challenge in network coding is the traffic overhead injected into the network. However specific analyses do not exist yet, neither simulative nor experimental. It can easily be envisioned that the next efforts on network coding will be both on simulation and experimentation to better understand real strengths and limits of this new technique. Other interesting challenges to be addressed in the framework of network coding are how to minimize the traffic injected into the network while still retaining sufficient redundancy in the encoding process, and how to efficiently deal with dynamically generated content.

6. Exercises

Exercise 1. Consider the encoding process of the Reed-Solomon codes. Given the Extension Field GF(2^8), which is the maximum possible size of the encoding matrix? Give a schematic representation of the matrix.

Exercise 2. Consider the bipartite graph of Fig. 13.

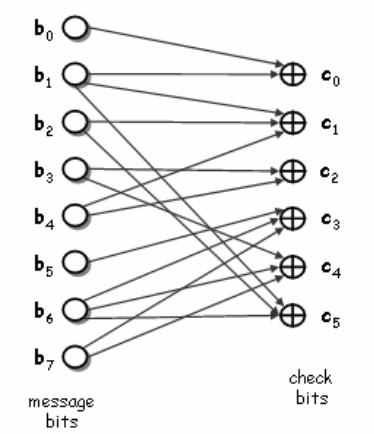

Fig. 13. First-level bipartite graph of a Tornado code.

Assume it is part of the multi-bipartite graph that describes a complete Tornado Code and that the decoding process has already produced the second-level check bits c_0, .., c_5 whose values follow.

$c_0 = 1$,
$c_1 = 1$,
$c_2 = 1$,
$c_3 = 1$,
$c_4 = 1$,
and $c_5 = 0$.

Assume also that the original message bits b_1 and b_4 are known because they have arrived incorrupted at destination.

$b_1 = 0$,
and $b_4 = 0$.

Complete the decoding process.

Exercise 3. Consider the graph of Fig. 14.

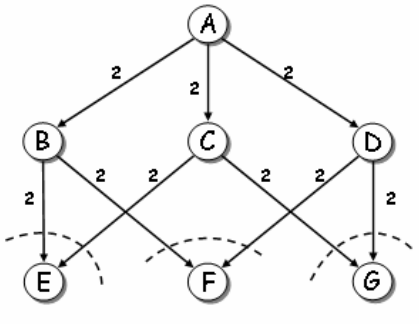

Fig. 14. Node A is the source node of the graph. Receiver nodes are nodes E, F, and G. Each edge has 2-bit capacity. Each destination node has the potential to receive 4 bits at the same time, two for each incoming edge.

The graph includes seven nodes. Node A is the only source whereas there are three destination nodes: E, F, and G. As is shown, each edge of the graph has a capacity of 2 bits. Hence, each of the destination nodes has the potential to receive four bits at the same time (see the max-flow min-cut theorem). Suppose the source node A has to send the four bits b_1, b_2, b_3, and b_4. Show a

possible transmission scheme of the four bits in the two cases *with* and *without* applying network coding. Then, compare the total number of transmissions needed in the two cases and evaluate the percent saving achieved when using network coding.
Use simple *xor* operations for the encoding.

Exercise 4. Consider the graph of Fig. 15.

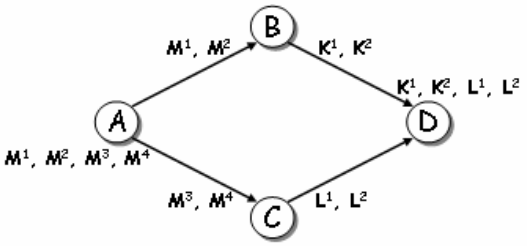

Fig. 15. Node A is the source node of four messages M_1, .., M_4 destined to node D. Intermediate nodes B and C receive (each) two out of the four source messages. They apply encoding over the received messages and forward the resultant encoded messages to node D.

Suppose Node A generates 4 messages destined to node D: M^1, M^2, M^3 and M^4. Suppose node B receives the messages M^1 and M^2 while node C receives the messages M^3 and M^4. Both nodes B and C apply a linear combination over the received messages and produce two encoded messages each. Node B produces messages K^1 and K^2, and node C produces messages L^1 and L^2. Finally, nodes B and C send out the encoded messages to the destination node D. Assume that the original messages M^1, M^2, M^3 and M^4 can be represented as 4-element vectors with the following expressions.

$$M^1 = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix},\ M^2 = \begin{pmatrix} 2 \\ 1 \\ 4 \\ 2 \end{pmatrix},\ M^3 = \begin{pmatrix} 3 \\ 1 \\ 1 \\ 3 \end{pmatrix}, \text{ and } M^4 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

Select, at will, the coefficients of the possible linear combinations that are applied at nodes B and C and work out possible values for the vectors L^1, L^2, K^1, and K^2. Show the form of the *encoding vectors* which are sent together with the *information vectors* and finally show the algebraic steps of the decoding process applied at node D, and demonstrate that the decoding process produces back the original messages M^1, M^2, M^3 and M^4.
For simplicity, work out additions and products in the field $(R,+,\cdot)$.

Exercise 5. Consider the graph of Fig. 16.

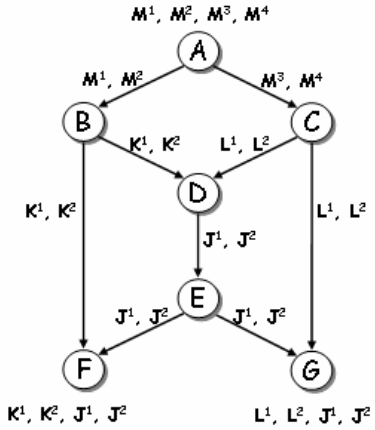

Fig. 16. Node A is the source node of four messages M_1, .., M_4 destined to both nodes F and G. Intermediate nodes B and C receive each one two out of the four source messages. They apply encoding over the received messages and then forward the resultant encoded messages to node D. Node D performs encoding over the received messages, too, while node E simply forwards the received messates to nodes F and G.

Similarly to the case of the Exercise 4, node A generates four messages: M^1, M^2, M^3 and M^4. Node B receives the messages M^1 and M^2 while node C receives the messages M^3 and M^4. Both node B and node C apply a linear combination over the received messages and produce two *encoded messages* each. Node B produces the messages K^1 and K^2, and node C produces the messages L^1 and L^2. Nodes B and C send out their encoded messages to node D.

Assume this time there are two destinations, namely node F and node G. After receiving the four encoded messages K^1, K^2, L^1 and L^2, node D applies a second-stage encoding and produces the messages J^1 and J^2 and then forwards them to node E. Node E sends out the same messages J^1 and J^2 it receives, without encoding. Finally the destination nodes F and G receive four messages each. Node F receives K^1, K^2, J^1 and J^2, while node G receives L^1, L^2, J^1 and J^2.

Starting from the same original and encoded vectors of the Exercise 4, i.e., M^1, M^2, M^3 and M^4, and K^1, K^2, L^1 and L^2, think of a possible linear combination that node D can apply to generate the encoded information vectors J^1 and J^2. Give the exact expression of the *encoding vectors* (vectors of coefficients) which have to accompany the *information vectors* J^1 and J^2 sent by node D. Finally, verify that the decoding process produces the exact original messages both at node F and node G.

For simplicity, work out additions and products in the field $(R,+,\cdot)$.

7. Appendix

7.1. Solutions of the proposed exercises

Solution 1. The encoding matrix of a Reed-Solomon code has the form of a Vandermond matrix. Assuming an *(n, k)* RS-code, the matrix looks as follows.

$$V = \begin{pmatrix} 1 & g_0 & g_0^2 & .. & g_0^{k-1} \\ 1 & g_1 & g_1^2 & .. & g_1^{k-1} \\ .. & .. & .. & & .. \\ .. & .. & .. & & .. \\ 1 & g_{n-1} & g_{n-1}^2 & .. & g_{n-1}^{k-1} \end{pmatrix}.$$

Where $g_i^j \in \left\{ x \mid x \in GF(p^r),\, x \neq 0 \right\}$, with *p* prime and *r > 0*. Furthermore, the following relation must hold true to have a non-singular matrix: $i \neq j \Rightarrow g_i \neq g_j,\ \forall\, g_i, g_j$.

The maximum number of columns of the Vandermonde matrix is constrained by the cardinality of the Extension Field $GF(p^r)$. Namely, it corresponds to the total number of non-null elements of the field, i.e., p^r*-1*. In our case, $p^r - 1 = 2^8 - 1 = 255$ and thus $n^{\max} = 255$. Moreover, since for (n, k) codes the relation $n \geq k$ holds, the maximum number of columns of the matrix is $k^{\max} = n^{\max} = 255$. Hence, $V^{\max}$ has size 255x255.

The encoding matrix can finally be extended with an identity matrix to generate systematic RS-codes. The size of the identity matrix is fixed given the number of columns of the Vandermonde matrix. Therefore the final matrix has the following form.

$$V = \begin{pmatrix} I_k \\ \hline 1 & g_0 & g_0^2 & .. & g_0^{k-1} \\ 1 & g_1 & g_1^2 & .. & g_1^{k-1} \\ .. & .. & .. & & .. \\ .. & .. & .. & & .. \\ 1 & g_{n-1} & g_{n-1}^2 & .. & g_{n-1}^{k-1} \end{pmatrix} = \begin{pmatrix} I_{n^{\max}} \\ \hline 1 & g_0 & g_0^2 & .. & g_0^{n^{\max}-1} \\ 1 & g_1 & g_1^2 & .. & g_1^{n^{\max}-1} \\ .. & .. & .. & & .. \\ .. & .. & .. & & .. \\ 1 & g_{n^{\max}-1} & g_{n^{\max}-1}^2 & .. & g_{n^{\max}-1}^{n^{\max}-1} \end{pmatrix}.$$

As is clear, the final size of the encoding matrix is 2(255)x255.

Solution 2. As is shown in the graph of Fig. 13, the following relations hold.

$$\begin{cases} c_0 = b_0 \oplus b_1 = 1 \\ c_1 = b_1 \oplus b_2 \oplus b_4 = 1 \\ c_2 = b_3 \oplus b_4 = 1 \\ c_3 = b_5 \oplus b_6 \oplus b_7 = 1 \\ c_4 = b_3 \oplus b_6 \oplus b_7 = 1 \\ c_5 = b_1 \oplus b_2 \oplus b_6 = 0 \end{cases} \Leftrightarrow \begin{cases} b_0 \oplus 0 = 1 \\ 0 \oplus b_2 \oplus 0 = 1 \\ b_3 \oplus 0 = 1 \\ b_5 \oplus b_6 \oplus b_7 = 1 \\ b_3 \oplus b_6 \oplus b_7 = 1 \\ 0 \oplus b_2 \oplus b_6 = 0 \end{cases} \Leftrightarrow \begin{cases} b_0 = 1 \\ b_2 = 1 \\ b_3 = 1 \\ b_5 \oplus b_6 \oplus b_7 = 1 \\ b_3 \oplus b_6 \oplus b_7 = 1 \\ b_2 \oplus b_6 = 0 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} b_0 = 1 \\ b_2 = 1 \\ b_3 = 1 \\ b_5 \oplus b_6 \oplus b_7 = 1 \\ 1 \oplus b_6 \oplus b_7 = 1 \\ 1 \oplus b_6 = 0 \end{cases} \Leftrightarrow \begin{cases} b_0 = 1 \\ b_2 = 1 \\ b_3 = 1 \\ b_5 \oplus b_6 \oplus b_7 = 1 \\ b_6 \oplus b_7 = 0 \\ b_6 = 1 \end{cases} \Leftrightarrow \begin{cases} b_0 = 1 \\ b_2 = 1 \\ b_3 = 1 \\ b_5 \oplus 0 = 1 \\ b_7 = 1 \\ b_6 = 1 \end{cases} \Leftrightarrow \begin{cases} b_0 = 1 \\ b_2 = 1 \\ b_3 = 1 \\ b_5 = 1 \\ b_7 = 1 \\ b_6 = 1 \end{cases}.$$

Hence the original information is $(b_7\ b_6 \ldots b_0) = (11101101)_2 = 237_{10}$.

Solution 3. Fig. 17 shows two possible transmission schemes. Scheme (a) shows the solution without network coding. Scheme (b) shows a possible solution with network coding. In scheme (a) two bits can be sent on the same link at the same time. To allow all the destination nodes (E, F, and G) receive all the bits, it is necessary that two consecutive transmissions are provided on the link AC. Hence, the total number of transmissions without network coding is 10.

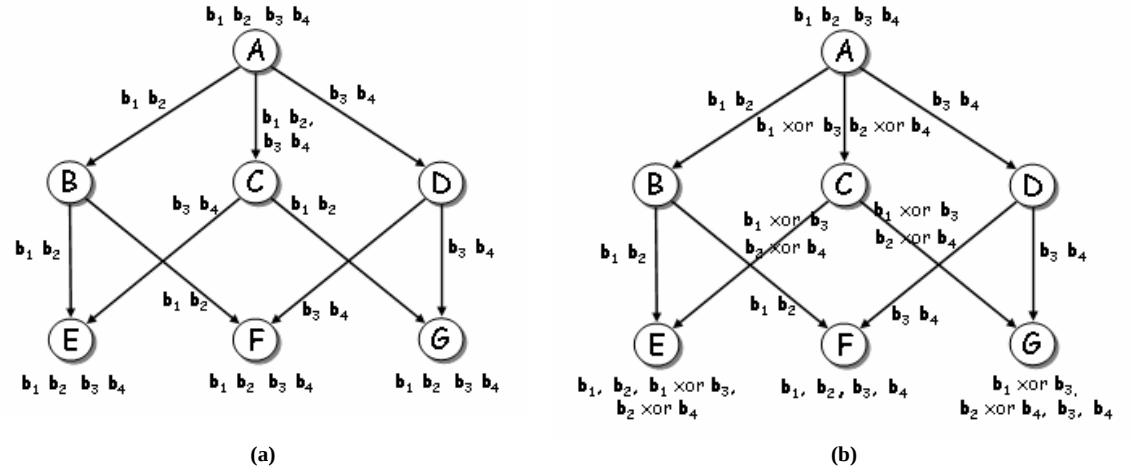

Fig. 17. (a) Possible transmission scheme of the four bits b_1, b_2, b_3, and b_4 from the node A to the nodes E, F, and G without using network coding. (b) Possible transmission scheme of the four bits b_1, b_2, b_3, and b_4 from the node A to the nodes E, F, and G using network coding.

In scheme (b) node A can avoid transmitting twice to node C. It transmits only once and transmits the two bits that result by xor-ing together nodes b_1 and b_3, and nodes b_2 and b_4. Node F receives the four original bits in plain. Nodes E and G can obtain all the information via decoding.

E: b_1; b_2; $b_3 = b_1$ xor $(b_1$ xor $b_3)$; $b_4 = b_2$ xor $(b_2$ xor $b_4)$.

F: b_3; b_4; $b_1 = b_3$ xor $(b_1$ xor $b_3)$; $b_2 = b_4$ xor $(b_2$ xor $b_4)$.

The total number of transmissions with network coding is 9. Hence one transmission has been avoided out of 10. The transmission saving has been 10%.

Solution 4. A possible solution for the Exercise 4 is as follows.

Node A sends out the following tuples: $<e_1, M^1>$, $<e_2, M^2>$, $<e_3, M^3>$, and $<e_4, M^4> = <(1, 0, 0, 0)^T, (1, 1, 2, 0)^T>$, $<(0, 1, 0, 0)^T, (2, 1, 4, 2)^T>$, $<(0, 0, 1, 0)^T, (3, 1, 1, 3)^T>$, and $<(0, 0, 0, 1)^T, (1, 0, 1, 0)^T>$.

Node B receives the tuples $<e_1, M^1>$ and $<e_2, M^2>$. Possible linear combinations to construct K^1 and K^2 follow.

$$K^1 = 2M^1 + M^2 = 2 \cdot \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \\ 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 8 \\ 2 \end{pmatrix};\ K^2 = -M^1 + M^2 = -\begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \\ 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 2 \end{pmatrix}.$$

Hence node B sends out the tuples: $< (2, 1, 0, 0)^T, (4, 3, 8, 2)^T >$ and
$< (-1, 1, 0, 0)^T, (1, 0, 2, 2)^T >$.

Node C receives the tuples <e_3, M^3> and <e_4, M^4>. Possible linear combinations to construct L^1 and L^2 follow.

$$L^1 = M^3 + M^4 = \begin{pmatrix} 3 \\ 1 \\ 1 \\ 3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 2 \\ 3 \end{pmatrix};\ L^2 = M^3 - M^4 = \begin{pmatrix} 3 \\ 1 \\ 1 \\ 3 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 0 \\ 3 \end{pmatrix}.$$

Hence node C sends out the tuples: $< (0, 0, 1, 1)^T, (4, 1, 2, 3)^T >$ and
$< (0, 0, 1, -1)^T, (2, 1, 0, 3)^T >$.

Finally, node D receives the following tuples: $< (2, 1, 0, 0)^T, (4, 3, 8, 2)^T >$,
$< (-1, 1, 0, 0)^T, (1, 0, 2, 2)^T >$, $< (0, 0, 1, 1)^T, (4, 1, 2, 3)^T >$, and
$< (0, 0, 1, -1)^T, (2, 1, 0, 3)^T >$.

The decoding process is as follows:

$$\left(\begin{array}{cccc|c} 2 & 1 & 0 & 0 & K^1 \\ -1 & 1 & 0 & 0 & K^2 \\ 0 & 0 & 1 & 1 & L^1 \\ 0 & 0 & 1 & -1 & L^2 \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|c} 3 & 0 & 0 & 0 & K^1 - K^2 \\ 0 & 3/2 & 0 & 0 & K^1/2 + K^2 \\ 0 & 0 & 2 & 0 & L^1 + L^2 \\ 0 & 0 & 0 & -2 & L^2 - L^1 \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & (K^1 - K^2)/3 \\ 0 & 1 & 0 & 0 & (K^1 + 2K^2)/3 \\ 0 & 0 & 1 & 0 & (L^1 + L^2)/2 \\ 0 & 0 & 0 & 1 & (L^1 - L^2)/2 \end{array}\right).$$

Summing up:

$$M^1 = \frac{(K^1 - K^2)}{3} = \frac{1}{3}\left[\begin{pmatrix} 4 \\ 3 \\ 8 \\ 2 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 2 \\ 2 \end{pmatrix}\right] = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix},$$

$$M^2 = \frac{(K^1 + 2K^2)}{3} = \frac{1}{3}\left[\begin{pmatrix} 4 \\ 3 \\ 8 \\ 2 \end{pmatrix} + 2 \cdot \begin{pmatrix} 1 \\ 0 \\ 2 \\ 2 \end{pmatrix}\right] = \begin{pmatrix} 2 \\ 1 \\ 4 \\ 2 \end{pmatrix},$$

$$M^3 = \frac{(L^1 + L^2)}{2} = \frac{1}{2}\left[\begin{pmatrix} 4 \\ 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \\ 0 \\ 3 \end{pmatrix}\right] = \begin{pmatrix} 3 \\ 1 \\ 1 \\ 3 \end{pmatrix},$$

$$M^4 = \frac{\left(L^1 - L^2\right)}{2} = \frac{1}{2}\left[\begin{pmatrix}4\\1\\2\\3\end{pmatrix} - \begin{pmatrix}2\\1\\0\\3\end{pmatrix}\right] = \begin{pmatrix}1\\0\\1\\0\end{pmatrix}.$$

Alternatively:

$$\left(\begin{array}{cccc|c} 2 & 1 & 0 & 0 & K^{1T} \\ -1 & 1 & 0 & 0 & K^{2T} \\ 0 & 0 & 1 & 1 & L^{1T} \\ 0 & 0 & 1 & -1 & L^{2T} \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|cccc} 2 & 1 & 0 & 0 & 4 & 3 & 8 & 2 \\ -1 & 1 & 0 & 0 & 1 & 0 & 2 & 2 \\ 0 & 0 & 1 & 1 & 4 & 1 & 2 & 3 \\ 0 & 0 & 1 & -1 & 2 & 1 & 0 & 3 \end{array}\right) \Leftrightarrow$$

$$\Leftrightarrow \left(\begin{array}{cccc|cccc} 3 & 0 & 0 & 0 & 3 & 3 & 6 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 4 & 2 \\ 0 & 0 & 2 & 0 & 6 & 2 & 2 & 6 \\ 0 & 0 & 0 & -1 & -1 & 0 & -1 & 0 \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 4 & 2 \\ 0 & 0 & 1 & 0 & 3 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}\right) =$$

$$= \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & M^{1T} \\ 0 & 1 & 0 & 0 & M^{2T} \\ 0 & 0 & 1 & 0 & M^{3T} \\ 0 & 0 & 0 & 1 & M^{4T} \end{array}\right).$$

Solution 5. A possible solution for the Exercise 5 is as follows.

Node D receives the following tuples:

$< g^1, K^1 > = < (2, 1, 0, 0)^T, (4, 3, 8, 2)^T >$, $< g^2, K^2 > = < (-1, 1, 0, 0)^T, (1, 0, 2, 2)^T >$,

$< t^1, L^1 > = < (0, 0, 1, 1)^T, (4, 1, 2, 3)^T >$, and $< t^2, L^2 > = < (0, 0, 1, -1)^T, (2, 1, 0, 3)^T >$.

Possible linear combinations for J^1 and J^2 follow.

$$J^1 = \begin{pmatrix}-2 & 2 & 1 & -1\end{pmatrix} \cdot \begin{pmatrix}K^1\\K^2\\L^1\\L^2\end{pmatrix} = -2K^1 + 2K^2 + L^1 - L^2 =$$

$$= -2 \cdot \begin{pmatrix}4\\3\\8\\2\end{pmatrix} + 2 \cdot \begin{pmatrix}1\\0\\2\\2\end{pmatrix} + \begin{pmatrix}4\\1\\2\\3\end{pmatrix} - \begin{pmatrix}2\\1\\0\\3\end{pmatrix} = \begin{pmatrix}-4\\-6\\-10\\0\end{pmatrix}.$$

$$J^2 = \begin{pmatrix}1 & 1 & -2 & 1\end{pmatrix} \cdot \begin{pmatrix}K^1\\K^2\\L^1\\L^2\end{pmatrix} = K^1 + K^2 - 2L^1 + L^2 = \begin{pmatrix}4\\3\\8\\2\end{pmatrix} + \begin{pmatrix}1\\0\\2\\2\end{pmatrix} - 2 \cdot \begin{pmatrix}4\\1\\2\\3\end{pmatrix} + \begin{pmatrix}2\\1\\0\\3\end{pmatrix} = \begin{pmatrix}-1\\2\\6\\1\end{pmatrix}.$$

The corresponding encoding vectors are obtained by the following expressions.

$$r^{1T} = \begin{pmatrix}-2 & 2 & 1 & -1\end{pmatrix} \cdot \begin{pmatrix}g^{1T}\\g^{2T}\\t^{1T}\\t^{2T}\end{pmatrix} = \begin{pmatrix}-2 & 2 & 1 & -1\end{pmatrix} \cdot \begin{pmatrix}2 & 1 & 0 & 0\\-1 & 1 & 0 & 0\\0 & 0 & 1 & 1\\0 & 0 & 1 & -1\end{pmatrix} = \begin{pmatrix}-6 & 0 & 0 & 2\end{pmatrix};$$

$$r^{2T} = \begin{pmatrix} 1 & 1 & -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} g^{1T} \\ g^{2T} \\ t^{1T} \\ t^{2T} \end{pmatrix} = \begin{pmatrix} 1 & 1 & -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & -1 & -3 \end{pmatrix}.$$

The tuples sent out by node D are as follows.

$< r^1, J^1 > = < (-6, 0, 0, 2)^T, (-4, -6, -10, 0)^T >,$

$< r^2, J^2 > = < (1, 2, -1, -3)^T, (-1, 2, 6, 1)^T >.$

Decoding process at node F:

$$\left(\begin{array}{c|c} t^{1T} & L^{1T} \\ t^{2T} & L^{2T} \\ r^{1T} & J^{1T} \\ r^{2T} & J^{2T} \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|cccc} 0 & 0 & 1 & 1 & 4 & 1 & 2 & 3 \\ 0 & 0 & 1 & -1 & 2 & 1 & 0 & 3 \\ -6 & 0 & 0 & 2 & -4 & -6 & -10 & 0 \\ 1 & 2 & -1 & -3 & -1 & 2 & 6 & 1 \end{array}\right) \Leftrightarrow$$

$$\left(\begin{array}{cccc|cccc} -6 & 0 & 0 & 2 & -4 & -6 & -10 & 0 \\ 1 & 2 & -1 & -3 & -1 & 2 & 6 & 1 \\ 0 & 0 & 2 & 0 & 6 & 2 & 2 & 6 \\ 0 & 0 & 0 & 2 & 2 & 0 & 2 & 0 \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|cccc} -6 & 0 & 0 & 0 & -6 & -6 & -12 & 0 \\ 1 & 2 & 0 & 0 & 5 & 3 & 10 & 4 \\ 0 & 0 & 1 & 0 & 3 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}\right) \Leftrightarrow$$

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 2 & 0 & 0 & 4 & 2 & 8 & 4 \\ 0 & 0 & 1 & 0 & 3 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 4 & 2 \\ 0 & 0 & 1 & 0 & 3 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}\right) = \left(\begin{array}{c|c} 1 & 0 & 0 & 0 & M^{1T} \\ 0 & 1 & 0 & 0 & M^{2T} \\ 0 & 0 & 1 & 0 & M^{3T} \\ 0 & 0 & 0 & 1 & M^{4T} \end{array}\right).$$

Decoding process at node G:

$$\left(\begin{array}{c|c} g^{1T} & K^{1T} \\ g^{2T} & K^{2T} \\ r^{1T} & J^{1T} \\ r^{2T} & J^{2T} \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|cccc} 2 & 1 & 0 & 0 & 4 & 3 & 8 & 2 \\ -1 & 1 & 0 & 0 & 1 & 0 & 2 & 2 \\ -6 & 0 & 0 & 2 & -4 & -6 & -10 & 0 \\ 1 & 2 & -1 & -3 & -1 & 2 & 6 & 1 \end{array}\right) \Leftrightarrow$$

$$\left(\begin{array}{cccc|cccc} 3 & 0 & 0 & 0 & 3 & 3 & 6 & 0 \\ 0 & 3/2 & 0 & 0 & 3 & 3/2 & 6 & 3 \\ 0 & 0 & 0 & 2 & 2 & 0 & 2 & 0 \\ 1 & 2 & -1 & -3 & -1 & 2 & 6 & 1 \end{array}\right) \Leftrightarrow \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 4 & 2 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 3 & -1 & -3 & 0 & 2 & 8 & 3 \end{array}\right) \Leftrightarrow$$

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 4 & 2 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 3 & 1 & 1 & 3 \end{array}\right) \Leftrightarrow \left(\begin{array}{c|c} 1 & 0 & 0 & 0 & M^{1T} \\ 0 & 1 & 0 & 0 & M^{2T} \\ 0 & 0 & 1 & 0 & M^{3T} \\ 0 & 0 & 0 & 1 & M^{4T} \end{array}\right).$$

8. References

[1] M. Conti, "Wireless Communications and Pervasive Technologies", *Chapter 4 of Environments: Technologies, Protocols and Applications*, Diane Cook and Sajal K. Das (Editors), John Wiley and Sons, 2004. pp. 63-99.

[2] I. Chlamtac, M. Conti, J. Liu, "Mobile Ad hoc Networking: Imperatives and Challenges", *Ad Hoc Networks Journal*, Vol.1 N.1 January-February-March, 2003.

[3] J.P. Macker, S. Corson, "Mobile Ad hoc Networks (MANET): Routing technology for dynamic, wireless networking", *in Mobile Ad hoc networking*, S. Basagni, M. Conti, S. Giordano, I. Stojmenovic (Editors), IEEE Press and John Wiley and Sons, Inc., New York, 2004.

[4] I. F. Akyildiz, M. C. Vuran, O. B. Akan, and W. Su, "Wireless Sensor Networks: A Survey Revisited", *Computer Networks* (2006).

[5] R. Bruno, M. Conti, and E. Gregori, "Mesh Networks: Commodity MultihopAd Hoc Networks", *IEEE Communications Magazine*, March 2005, pp.123-13.

[6] M. Conti, "Principles and Applications of Ad Hoc and Sensor Networks", *The Handbook of Computer Networks*, Hossein Bidgoli (Editor), John Wiley & Sons, to appear.

[7] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad hoc Networks", *IEEE Communications Magazine*, to appear.

[8] L. Rizzo and L. Vicisano, "RMDP: an FEC-based Reliable Multicast protocol for wireless environments", *in ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 2, pp. 23-31, 1998.

[9] S. Kim, R. Fonseca, and D. Culler, "Reliable Transfer on Wireless Sensor Networks", *in Proceedings of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (IEEE SECON 2004)*, October 4-7, 2004.

[10] P. Karlsson, L. Öberg, and Y. Xu, "An Address Coding Scheme for Wireless Sensor Networks", *in Proceedings of the 5th Scandinavian Workshop on Wireless Ad-hoc Networks (ADHOC ´05)*, Stockholm, May 3-4, 2005.

[11] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes", *in Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN 2005)*, April 25-27, Sunset Village, UCLA, Los Angeles, CA.

[12] H. Dong, I. D. Chakeres, A. Gersho, E. M. B.-R. U. Madhow, and J. D. Gibson, "Speech Coding for Mobile Ad hoc Networks", *in Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 2003.

[13] S. Deb, M. Effros, T. Ho, D. R. Karger, R. Koetter, D. S. Lun, M. Médard, and N. Ratnakar, "Network coding for wireless applications: A brief tutorial", *in Proceedings of the International Workshop on Wireless Ad-hoc Networks (IWWAN)*, London, UK, May 23-26, 2005.

[14] Y. Wu, P. A. Chou, and S.-Y. Kung, "Minimum-Energy Multicast in Mobile Ad Hoc Networks using Network Coding", *IEEE Transactions on Communications*, Volume 53, Issue 11, Nov. 2005 Page(s): 1906 – 1918.

[15] Y. Wu, P. A. Chou, and S.-Y. Kung, "Information Exchange in Wireless Networks with Network Codign and Physical-layer Broadcast" *in Proceedings of the 39th Annual Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, March 16-18, 2005.

[16] J. Widmer, C. Fragouli, and J.-Y. Le Boudec, "Low-complexity energy-efficient broadcasting in wireless ad hoc networks using network coding", *in Proceedings of the 1st Workshop on Network Coding, Theory, and Applications*, April 2005.

[17] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in The Air: Practical Wireless Network Coding", *in Proceedings of the ACM SIGCOMM 2006*, Pisa, Italy, Sep. 11-15, 2006.

[18] A. A. Hamra, C. Barakat, and T. Turletti, "Network Coding for Wireless Mesh Networks: A Case Study", *in Proceedings of the 2nd IEEE International Conference on the World of Wireless, Mobile and Multimedia Networks (WoWMoM2006)*, Niagara-Falls / Buffalo, NY, June 26-29, 2006.

[19] J. Widmer and J.-Y. Le Boudec, "Network Coding for Efficient Communication in Extreme Networks", *in Proceedings of the ACM SIGCOMM 2005 Workshop on delay tolerant networks*, Philadelphia, PA, USA, August 22–26, 2005.

[20] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla, "CodeTorrent: Content Distribution using Network Coding in VANETs", *in Proceedings of the 1st International ACM Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (ACM MobiShare) in conjunction with ACM Mobicom 2006*, Los Angeles, CA, USA, September 25, 2006.

[21] C.E. Shannon, "A Mathematical Theory of Communication", *in The Bell System Technical Journal*, Vol. 27, pp. 379-423, 623-656, July, October, 1948.

[22] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols", *in ACM Computer Communication Review*, Vol. 27, n. 2, April 1997, pp. 24-36.

[23] L. Rizzo, "On the feasibility of software FEC", *in DEIT Technical Report LR-970131*, available as http://www.iet.unipi.it/luigi/softfec.ps, January 1997.

[24] J.B. Byers, M. Luby, and M. Mitzenmacher, "A Digital Fountain Approach to Asynchronous Reliable Multicast", *in IEEE Journal on Selected Areas in Communications*, Vol. 20, No. 8, Obctober 2002.

[25] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields", *Journal of the Society for Industrial and Applied Mathematics*, 8:300-304, June 1960.

[26] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical Loss-Resilient Codes", *in Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 150-159.

[27] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes", *IEEE Transactions on Information Theory*, 47(2):569-584, February 2001.

[28] M. Luby, "LT codes", *in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 271-282, 2002.

[29] A. Shokrollahi, “Raptor codes”, *in Proceedings of the IEEE International Symposium on Information Theory (ISIT 2004)*, Chicago Downtown Marriott, Chicago, IL USA, June 27 - July 2, 2004. Preprint available at http://algo.epfl.ch/pubs/raptor.pdf.
[30] A. Shokrollahi, S. Lassen, and M. Luby, “Multi-stage code generator and decoder for communication systems”, *U.S. Patent Application #20030058958*.
[31] J. H. Lint, “Introduction to the Coding Theory”, Springer-Verlag, Berlin, 1982.
[32] S. Lin, “An Introduction to Error-Correcting Codes”, Prectice-Hall, Inc., Englewood Cliffs, New Jersey, 1970.
[33] R. Fonseca, D. Culler, S. Ratnasamy, S. Shenker, and I. Stoica, “Beacon vector routing: Scalable point-to-point routing in wireless sensornets”, under submission.
[34] R. G. Gallagar, “Low Density Parity-Check Codes”, Cambridge MA: MIT Press, 1963.
[35] T. Richardson, M. A. Shokrollahi, and R. Urbanke, “Design of capacity approaching irregular low-density parity-check codes”, *in IEEE Transactions on Information Theory*, Vol. 47, pp. 619–637, Feb. 2000.
[36] S. Robinson, “Beyond Reed–Solomon: New Codes for Internet Multicasting Drive Silicon Valley Start-up”, *from SIAM News*, Volume 35, Number 4, May 3, 2002.
[37] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow”, *IEEE Transactions on Information Theory*, vol. 46, pp. 1204-1216, July 2000. http://personal.ie.cuhk.edu.hk/~pwkwok4/Yeung/1.pdf.
[38] R. Koetter and M. Médard, “Beyond Routing: An Algebraic Approach to Network Coding”, *in Proceedings of INFOCOM 2002*.
[39] T. Noguchi, T. Matsuda, and M. Yamamoto, “Performance Evaluation of New Multicast Architecture with Network Coding”, *IEICE Trans. Comm.*, June, 2003.
[40] D. S. Lun, M. Médard, R. Koetter, “Network Coding for Efficient Wireless Unicast”, *in Proceedings of the IEEE International Zurich Seminar on Communications*, ETH Zurich, Switzerland, February 22-24, 2006.
[41] C. Chekuri, C. Fragouli, and E. Soljanin, “On average throughput and alphabet size in network coding”, *accepted to IEEE Transactions on Information Theory*.
[42] Z. Li and B. Li, “Network coding in undirected networks”, *CISS*, 2004.
[43] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, “Network Coding: An Instant Primer”, *in ACM Computer Communication Review*, Vol. 36, Nr. 1, pp. 63-68, 2006.
[44] S.-Y. R. Li, R. W. Yeung, and N. Cai, “Linear network coding”, *in IEEE Transactions on Information Theory*, 49(2):371–381, 2003.
[45] P. Sanders, S. Egner, and L. Tolhuizen, “Polynomial time algorithms for network information flow”, *in Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures*, 2003.
[46] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, “The benefits of coding over routing in a randomized setting”, *in Proceedings of the IEEE International Symposium on Information Theory*, June 2003.
[47] T. Ho, M. Médard, J. Shi, M. Effros, and D. R. Karger, “On randomized network coding”, *in Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing*, October 2003.
[48] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, “The benefits of coding over routing in a randomized setting”, *in Proceedings of the IEEE International Symposium on Information Theory (ISIT 2003)*, Yokohama, Japan, June 29 - July 4, 2003.
[49] C. Fragouli and E. Soljanin, “Decentralized network coding”, *Information Theory Workshop*, Oct. 2004.
[50] C. Gkantsidis and P. Rodriguez, “Network coding for large scale content distribution”, *in INFOCOM*, Miami, FL, Mar. 2005.
[51] Avalanche: File swarming with network coding. http://research.microsoft.com/pablo/avalanche.aspx.