Optimization of Random Linear Network Coding protocols for multipath communications in heterogeneous networks

April 25, 2019

Matriculation number: 4812346
Processing period: 17.09.2018-15.04.2019
Reviewer: Prof.Dr.-Ing.Dr h.c. Fitzek

Bachelor Thesis

Clara Costa Sala

supervised by
Roberto Torre Arranz

Fakültat Elektrotechnik und Informationstechnik, Institut für Nachrichtentechnik
Deutsche Telekom Chair of Communication Networks

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without acknowledgement in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been acknowledged.

Dresden, 15. April, 2019

....
Surname and name of the student

ALL copies or translated text must be market AND referenced

"All models are wrong, but some are useful."[1]

ALL adopted ideas must be referenced.

The new approach works best in meshed networks. [2]

Missing references or marks are counted as plagiarism and followed by a mark "Failed" / 5.

....

Student's signature *Supervisor's signature*

Abstract

In the next upcoming mobile generation era (5G), the demands for high-data rate transmissions is predicted to grow considerably. Among new studies focused on how to support such demands, Random Linear Network Coding (RLNC) has taken an important place for improving the performance of the network. In this thesis, the problem, arising from combining an RLNC protocol with multi-path communications, is considered. This thesis seeks to improve an RLNC protocol aimed for transmissions in multipath communications by enhancing an implementation of an RLNC protocol called *PACEMG* (PACE multi-generation). PACEMG is a protocol created in the Deutsche Telekom Chair of Communication Networks (TU-Dresden) which aims to support non-sequential arrival of packets from different generations by managing multiple generations simultaneously. This disordered arrival of information is highly probable to happen in multipath communications. Nodes that execute Random Linear Network Coding can perform three different activities: encode, recode and decode, hence, RLNC protocols are divided into three main parts that go by the name of encoder, decoder and recoder. The main objective of this thesis is to improve the PACEMG protocol by developing the recoder. Furthermore, some aspects of the encoder and decoder will be improved. Subsequently, its performance will be examined and some features will be compared to PACEMG previous version *PACE*[1]. A program to perform an emulation of a network is required so as to test the behaviour of the protocol. This thesis shows an improved RLNC approach to widen the use of Network Coding in communications, particularly multi-path communications. The advantages and constraints of using certain configurations of the recoder, precisely, four different configurations, will be presented. Finally, conclusions are drawn concerning the coefficients used in the coders with an approach that tries to overcome them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Network Background

The number of mobile wireless devices has been increasing in the past few years and it is expected to have an impressive growth in the next ones. It's use is augmenting, the demands of users to be constantly connected is increasing in such a way that it is estimated that the number of mobile-connected devices will reach 12.3 billion by 2022 [3] and, with it, the mobile data traffic. Yet the technology is being used nowadays won't be enough to support this amount of traffic. Wireless communications are of great importance between devices, however, they are unreliable in the presence of fading and interference, making this method less adequate in areas with a lot of traffic, more, if the communication growth that it is expeted to be faced is taken into consideration. This concern led to different studies on how to improve communications so that reliability and network performance is improved. With the arrival of the latest mobile generations (3G, 4G) these goals have been getting closer. But as the volume of mobile devices grow, number of links, one-to-one, one-to-many or many-to-many, are getting denser. Yet current deployment technology will not be able to scale such data traffic. Moreover, it is foreseen that the 79% of the world's mobile data traffic will be video by 2022 [3], the wireless high data rate demands will come with it, as well as bandwidth requirements.

1.2 Motivation

Reliability in the delivery of information (for instance in video transmissions) on wireless channels is often contradictory to a low delay transmission, and with high rate transmissions, the usage of retransmissions is undesired. With the use of Forward Error Correction (FEC) mechanisms, these features can be fixed. Random Linear Network Coding (RLNC) has been proven to be an effective FEC method to transmit data by having a good potential for the enhancement of network capacity, reliability [4, 5] and efficiency [6–8]. It promises to offer benefits along very diverse dimensions of communication networks, such as throughput, wireless resources [6], security [9, 10], complexity [11, 12], and resilience to link failures [13, 14].
Taking into consideration that in this upcoming technology generation everyone's connectivity should be supported, and this requires plenty of deployments which take time and resources, with RLNC by itself would not suffice to support the data traffic volume that is expected. Not only should the connectivity be provided, but also a high resilience and low latency connection too. To start with, using centralized communications can lead to a pooe service as a result of its susceptibility to failures. Moreover, it has a very limited scalability which may not turn out to be a cost-effective solution in the long run (with multiple devices connected to it). For that reason, decentralized networks have been considered to take place in communications. A new technology based on the concept of decentralization which consists of the use of the devices in it (routers, computers, mobile phones and other terminals) in order to directly interconnect one another wirelessly without following a predetermined hierarchical tree-topology, would help supporting a large amount of communications [15]. Nodes are connected to the others, as long as the *others* stay located within the range of the connection conditions, and share resources opportunistically. In other words, all the devices in a network are directly connected to the other devices that can be reached and, only one node needs to be physically wired to an access point, which allows devices to communicate with other nodes without being routed through a central switch point. This decentralized technology called Wireless Mesh Networks is characterized by dynamic self-organization,

self-configuration and self-healing to enable quick deployment, easy maintenance, low cost, high scalability and reliable services, as well as enhancing network capacity [16], connectivity and resilience [13]. An upcoming routing model of broadcast-channel networks called Opportunistic Routing (OpR) can be used in decentralized networks. By using OpR the data don't have a precomputed route to reach the destination but nodes decide dynamically whether or not to transmit certain packets to some other nodes. What is more, the combination of OpR with the aforementioned RLNC has shown improvements in the throughput, reliability and resilience of the network [17–24].

Broadcasting together with Network Coding is considered to substantially improve the throughput of the network [25]. However, the whys and wherefores broadcasting is not proposed, is because reliability can be guaranteed employing a feedback mechanism in single path communications to ensure the correct arrival of the packets if necessary, which it is not appropriate employing with multicasting. Furthermore, even though NC has been commonly researched in wireless environments relying on the broadcast of a single packet to multiple receivers to achieve high rates compared to a non-coding scheme [26], it has neither collision detection nor avoidance mechanism, which makes it not convenient to be used in environments where the collision probability is high. Consequently, as unicast RLNC has also proved to be beneficial [27], it could be used together with multipath communications.

Using a decentralized network with a single-path communication (without OpR) may end up being not so well suited for some applications due to its vulnerability to failures, especially when a node fails, all paths and nodes depending on *that* one node are temporarily inoperative forcing the need to update all the routes in the entire network that involve *that* node. The use of multi-path protocols, which means the use of diverse good paths instead of the optimal path in communications [28], has been recognized as an important feature in networks for being beneficial in some aspects such as fault tolerance, increased bandwidth, or improved security. By having redundant information routed to the destination via alternative paths the probability that communication is disrupted in case of link failure is reduced, the burden of links is balanced, effective bandwidth can be aggregated and end-to-end delay may also be reduced as a direct result of larger bandwidth [29]. Therefore, combining the use of Random Linear Network Coding (RLNC) with multipath unicast communications in a decentralized network can significantly improve the performance of the network [30] .

1.3 Problem description

The main disadvantage associated with the use of multi-path communications is the caused effect of non-sequential arrival of the information at the destination. Given that the characteristics of the links can vary very rapidly, information can take different routes causing diverse times of travel between packets. Introducing RLNC into multi-path communications would mean that nodes could receive packets from different generations consecutively. Figure 1.1 represents this scenario, where a packet from generation 2 arrives before a packet from generation 1 because they use different routes.

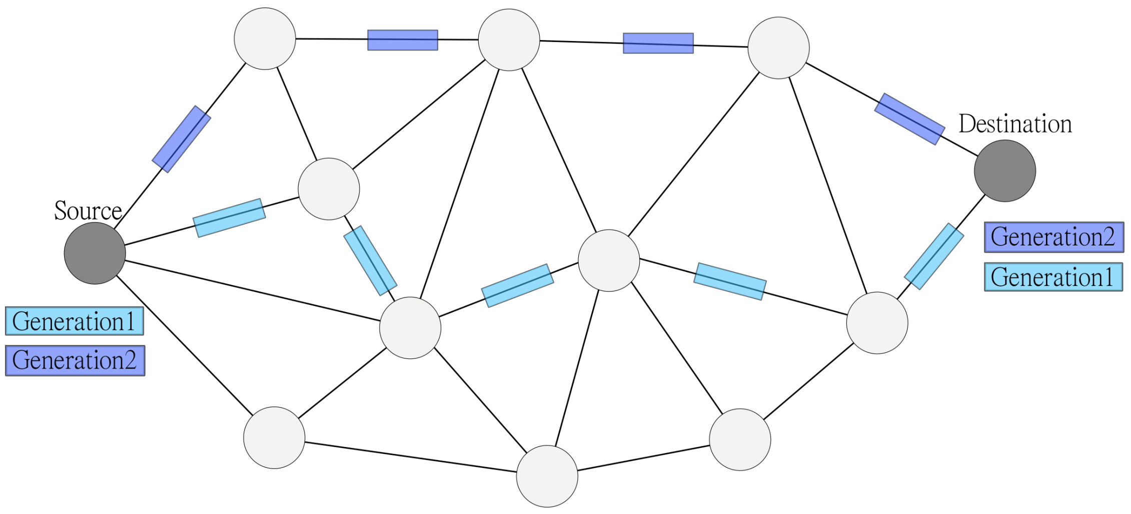


Figure 1.1: Multi-path scenario with one source and one destination.

There are two packets in the network sent from the source node: one from generation 1 and another from generation 2. Both packets use different paths and, the last sent packet is the first to arrive.

Some RLNC protocols assume that the communications between two devices will be in-order-delay and therefore, they can only manage one generation [1, 11] and do not take into consideration a scenario where different delay values for consecutive packets is possible. Hence, a low-delay RLNC protocol is needed optimized when the delay variance of the paths (which can be called *jitter*) is so diverse that it needs to handle different generations simultaneously, so that it can be implemented together with multi-path communications. In this thesis a RLNC that fulfills this characteristic proposed by 5G lab TU-Dresden is presented by the name of PACEMG (PACE multi-generation), the next version of Pace [1]. This protocol is formed of an encoder and a decoder. It is needed an element in this protocol called *recoder*, that is able to encode and decode.

1.4 Related Work

Past studies have investigated methods to improve Network Coding. Reliable Low-delay Network Coding have shown general benefits [31–33]. A deeper study for low latency applications focuses on the impact of generation and symbol sizes on latency for encoding with Random Linear Network Coding (RLNC) [34]. The study of different network codes for low-delay communication is shown in [35–39] and for low decoding complexity [14, 40]. Network coding was proposed for peer-to-peer content distribution systems [41] where random linear operations over packets are performed to improve downloading. Also in [42–44] Network Coding is proposed for peer-to-peer networks. The proposal and investigation of instant decodable network codes [45–51] show improvements in the latency. An approach called online codes (explained in Section 2.4.3) is studied in [52–58]. The study of Block codes is done in [59, 60], in contrast, in this thesis it is considered the systematic generation based form of RLNC. The throughput-delay performance of block based RLNC in multicast and unicast systems is done in [31–33, 61–71]. The impact of the disparity delay between packets when using NC is investigated in [72, 73]. Latest RLNC investigations focus on how to adapt RLNC to a network [74, 75] and some RLNC approaches are presented [30, 76–79]. Among different approaches, PACE [1] has shown to improve the delay in communications and will be considered in this thesis.

1.5 Unsolved gap

As PACE has proved to be a short delay protocol for RLNC [78], multi-generation based PACE aims to be used in multi-path communication for a short-delay communication. This protocol focus on managing data grom different generations in oder to allow a disordered arrival of information. The protocol is unfinished as it can only work in a single hop tranmisssion because it only has the encoder and decoder (the coders situated at the extremes of a communication).

1.6 Contribution

A protocol based on Random Linear Network Coding for multi-path communications developed in the 5G lab TU-Dresden that goes by the name of *Pacemg* is presented and improved in this thesis. This RLNC protocol deals with the arrival of disorganized packets which is the most common event to happen in multipath communications. When packets are sent through different paths at the same time in an heterogeneous network, packets will use different routes to get to the destinations and every route will have different characteristics which will make the packet arrive with another delay as another packet. This fact can make information arrive earlier than other information that was sent before. The multi-generation PACE protocol is not finished as the *recoder* member is of great importance to have a multi-hop communication. This member, which will be developed in this thesis is works as an encoder and decoder together and is shown to improve the previous protocol as it can be implemented in multi-hop communications.

1.7 Thesis overview

The remaining of this thesis is organized as follows: In Chapter 2 a review of Network Coding history is done, followed by a theoretical basis of NC and an improved method called Random Linear Network Coding (RLNC). In the subsequent sections, some important RLNC Codes are presented. Chapter 3 describes the previous version of the protocol that will be used (PACE) and the protocol PACEMG. Previous to that the library which these two protocols depend on (Section 3.1) are introduced. The next sections in the chapter explain, in addition to the PACE and PACEMG functionality, its mode of functioning together with the libraries. The implementation of this thesis is presented in Chapter 4. Four generated versions of the Recoder developed in this thesis that later are tested are explained in Section 4.1. In the following Chapter 5, the simulation to prove the good performance of the protocol is presented. The results of the simulations are shown with some conclusions in Sections 5.1 and 5.2

Chapter 2

Foundations of Network Coding NC

2.1 A brief history of network coding

Over recent years several approaches have been taken to find a method to solve the failures during transmissions in a way that errors can be corrected at sink nodes, which go by the name of Error Detection and Error Correction Codes. These codes consist of an algorithm to express a sequence of numbers such that any errors which are introduced can be detected and corrected (within certain limitations) based on the remaining numbers. While error detection is helpful in data transmissions, error correction is essential for a successful communication. With the insertion of redundancy (redundant bits), a receiver can detect some errors and. Additionally, these errors can be corrected to return a transmitted message to its original form. The first to introduce these codes were Cai and Yeung in [80, 81]. The concept of Network Coding it is said to have officially appeared when a landmark paper by Ahlswede et al. [2] was published in 2000. The introduction of this concept is only explained by a theoretical perspective, it is proved that good (informative) codes exist and the network capacity can be increased compared to employing routing alone, although they do not describe a method for designing them. Since then, the study of Network Coding has been explored by many universities in the field of Information theory. In 2003, Li et al. published a paper [82] that showed that network coding for multicast networks can rely on linear mathemati-

cal functions, involving only addition and multiplication, which reduced the complexity of designing codes. They presented a solution, the butterfly network, which will be explained in section 2.2, suggesting that the max-flow bound (multicast maximum capacity) could be achieved by linear NC. The approach to network coding presented a gain in terms of throughput. Subsequently, Koetter and Medard proposed an algebraic solution for network coding in [83] where they developed a sufficient condition for linear inter-session network coding optimization. They showed that any multicast network problem has a linear solution over a Galois field $GF(q)$. In other words, the linear codes can bring the multicast maximum capacity using this field size in multicast communications.
The concept of randomized network coding was first described in [84], which considering linearly independent or linear correlated sources on acyclic delay-free network, set an upper bound on error probability that decreases exponentially with the length of the codes. In order to implement NC, Ho et al. introduced in [85] the method of random linear network coding in multicast networks, which is explained in Section 2.3. They showed that by using randomized mappings in the inputs in multicast scenarios, it would help increase the system performance, it could increase the capacity of the network and robustness to network changes or link failures.

The first approaches of a practical randomized network coding were Block based RLNC. It has a great performance and its usefulness is of importance among researchers but, although RLNC can provide improvements in terms of throughout [59, 84], it can have some drawbacks considering the complexity when decoding and the reception delay. It is one of the strongest arguments against the use of RLNC, its decoding complexity can be roughly estimated as $\mathcal{O}(K^3)$, K being the number of symbols per block, which is notably high in comparison to other sparse end-to-end erasure correcting code schemes, such as LT [86] and Raptor Codes [87]. Even though they reduce the decoding load on the receiver, they do it at the cost of introducing additional, non-negligible delay. LT and Raptor Codes have some limitations especially when the packets need to go through several nodes (e.g. Mesh Networks) recoding at intermediate nodes it is not considered, which is one of the most relevant features of the RLNC

scheme. Investigation focused on the decoding complexity [88] ended up with a solution to the above problem in RLNC. This solution is explored in [59] by starting encoding on a partial block and adding packets to the generation as they arrive. By performing systematic RLNC [89] packet-delay is reduced and so is the decoding complexity.
In the rest of the chapter, NC main concepts will be introduced.

2.2 The basics of Network Coding (NC)

The theory of Network Coding (NC) have been given various definitions since its beginning. The underlying idea of network coding is nothing but carrying out operations on the contents of the packets that are intended to be sent. In the seminal paper Ahlswede et al. [2] they say that they "refer to coding at a node in a network as network coding". Therefore, Network Coding can be identified as a combination of several packets into a single packet, done in a node of a network, in order to create a *coded packet* with the same size as the source ones. NC, instead of considering a node in the network as a simple element for information flow relay, such as routers that simply store and forward data (unmodified data), nodes, compute and forward (by combining input data and forwarding coded data). Various theoretical and empirical studies suggest that significant gains can be obtained by using network coding in multi-hop wireless networks, also for serving multicast sessions (e.g. [90–92]). Network coding improves the throughput as packet transmission are more efficient [2] noting that the information of a packet that has been lost during a transmission can be recovered when receiving a combination of that packet with others already received. Network Coding also has the characteristic of bringing more security and complexity to the network [93]. Two methods of network coding can be identified: (a) intra-session coding, (b) inter-session coding.

(a) Intra-session NC: In this approach the coded packets created are the combination of the packets addressed to the same destination. Intermediate nodes do not decode the packets, the destination does. The coding of data is done to the same flow but can arrive from different nodes.

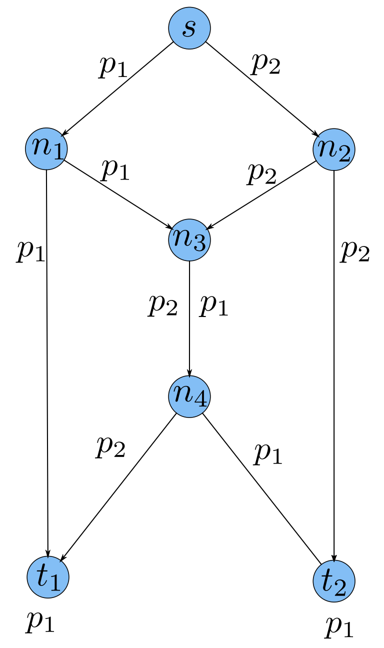
(b) Inter-session NC: Instead of combining input packets directed to the same destination, this approach combines all input packets in the node. It combines packets from different flows in order to improve the throughput.

In the rest of this section, two well known examples to demonstrate the basic principle of network coding and, its potential to improve throughput and achieve the capacity of a network are presented. The butterfly example (Section 2.2) is an example of an inter-session multicast wired NC communication and, the wireless example (Section 2.2) shows an inter-session wireless multicast NC scenario.

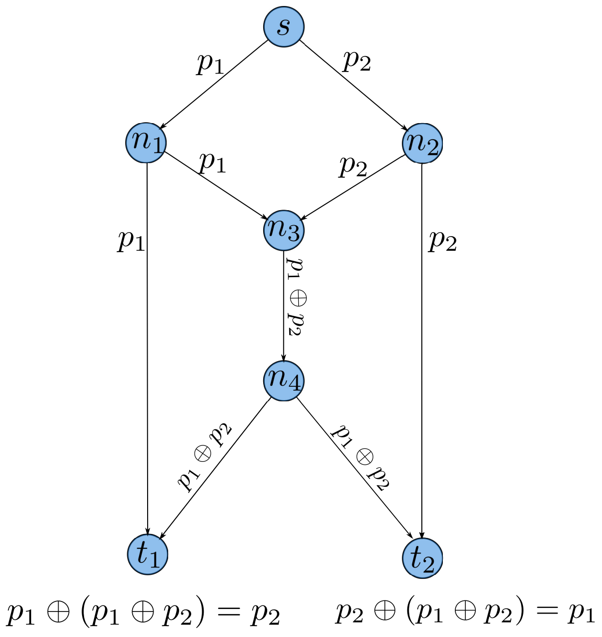
NC in a Butterfly Network

In order to explain the concept of network coding and show a simple scene where network coding increases the throughput, Ahlswede et al. used what is called a *butterfly network*, which later became very popular by the name of the *butterfly example*. Their example represents a multicast communication with a single source (s) and two destination nodes (t_1, t_2), assuming error-free and erasure-free links and each link carries a single unit. Figure 2.1 represents two scenarios where, in both of them, the source node s sends two packets p_1 and p_2 addressed to the destinations nodes t_1 and t_2.

Nodes n_1 and n_2 receive these packets and forward them to both n_3 and t_1 or t_2. When n_3 receives two different packets, it combines them by performing a XOR operation, $p_1 \oplus p_2$, and sends it to node n_4 which sends the same packet to the destination nodes. t_1 and t_2 then, using the same operation XOR but with p_1 and p_2 respectively, they can get p_2 and p_1 respectively. In other words, t_1, that received p_1 directly from node 1 n_1, can get p_2 by performing $p_1 \oplus (p_1 \oplus p_2)$, and so t_2 can obtain p_1 using the received packet p_2, $p_2 \oplus (p_1 \oplus p_2)$.
It can easily be inferred that the conventional approach (Figure 2.1a) requires 10 transmissions, while the approach in Figure 2.1b requires 9. The coding gain is then 10/9 in



(a) Without NC



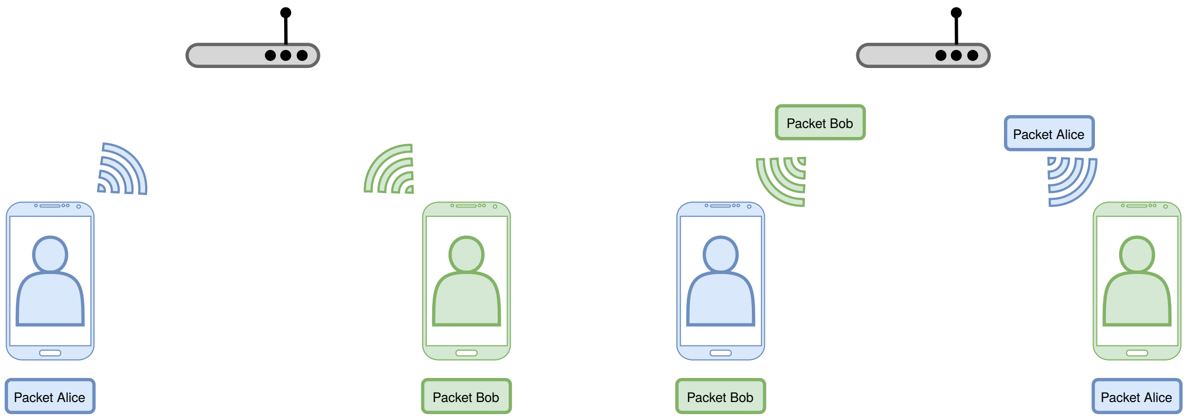
(b) Using NC

Figure 2.1: Butterfly example presented in [2].

the wired network. As illustrated in Figure 2.1, network coding can increase throughput for multicast in wired network. With this example Ahlswede et al. showed that network coding can help to send the data traffic at the same rate as the *min-cut theorem* [94] between the sender and the receiver, noted that nodes deliver combinations of the original data and senders and receivers don't need to track every individual packet in order to know if a packet has been lost, but rather focus on gathering enough independent linear combinations packets to be able to decode them into the original ones.

NC in a Wireless Network

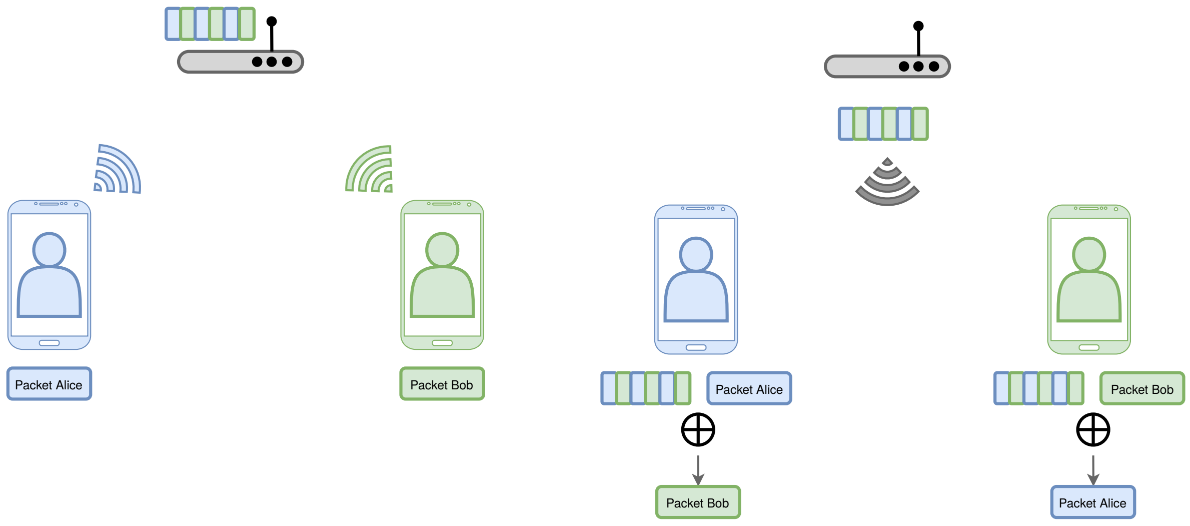
Another example where we can see the use of NC was presented in [6]. Some applications of network coding in wireless environments to address multiple unicast flows showed negative theoretical results [90, 95, 96] as the intermediate nodes mix packets from different flows without being concerned whether their paths separate. Katti et al. showed a new forwarding architecture named COPE that contributed with the use of network coding in unicast wireless communications using OpR. In the paper [6], they present a scenario as the one in Figure 2.2a and Figure 2.3a to show the benefits of network coding in a wireless network.



Alice and Bob send each other a packet. They send them to the router and the router forwards every packet to the destination, resulting in a communication of 4 transmissions.

(a) Wireless transmission without using NC presented in [6].

As illustrated, there are two wireless devices called *Alice* and *Bob* and a router. *Alice* and *Bob* want to interchange packets through the router, therefore, *Alice* sends the blue packet and *Bob* sends the green one to the router. In Figure 2.2a it is shown a scenario without network coding, where the router transmits the green packet to *Alice* and the blue one to *Bob*. Alternatively, in Figure 2.3a, the transmission of the packets is done by using network coding, thereby, the process where the router had to do two different transmissions to send the two packets in the not-coding scenario is changed for a one-transmission process. The router combines (by performing an XOR operation)



Alice and Bob send a packet to each other and the router, instead of forwarding Alice's packet to Bob and Bob's to Alice, combines both and broadcasts it. Alice will be able to get Bob's packet by performing an XOR to the received packet with hers, and so will Bob.

(a) Wireless communication using NC.

the two packets and broadcasts the result packet, which, is acquired by both devices. As *Alice* has the other packet (blue) she can decode the coded one by performing XOR with her packet. *Bob* can also do the same with his packet. In the picture the coded packet is the one with two colors, representing the combination of the blue and green packets. This example is a clear demonstration of the increase in the throughput in a wireless communication since, instead of taking 4 transmissions (when not using network coding), it takes 3 transmissions for the two devices to interchange information. One transmission has been saved which, could be used to send another.

When Koetter and Medard proposed to solve the network coding problem algebraically in [83], they explained the case of a multicast session where the source node transfers data to several destination nodes. Referencing this example, Koetter and Medard demonstrated that the single-source multicast capacity of a network could be achievable when using network coding over a finite field $\mathbb{F}_q$, with a $q > |T|$, $|T|$ being the number of sink nodes [83]. The question whether a coding solution can be found in a computationally efficient manner is solved in [92]. Jaggi et al. proposes a polynomial-time solution to this problem. However, these coefficients are chosen after the determination of an algebraic system, which adds up to finding a set of solutions of a system of polynomial equations that assigns the coefficients values. In other words, the solution given is found using knowledge of the entire network topology which gives some limitations when applying NC in real communications. The knowledge of an entire network is not always possible and makes the NC problem complex if a linear coding solution has to be found. Moreover, the solution is vulnerable to changes in the network, although if the topology is not to be re-designed, there are some approaches on how to assign encoding equations [93]. Nevertheless, topologies tend to change and, with Random linear network coding (RLNC) (Section 2.3) these constraints are overcome.

2.3 Random Linear Network coding (RLNC)

The particular form of network coding called random linear network coding (RLNC) a class of network coding introduced in [85], lies on the concept of a "random linear mapping of input to output data symbols over a finite field". The node receives incoming packets and then, it encodes the source packets into a network-coded packet using, instead of predefined, random coefficients. It generates a local coding vector randomly and locates the coding information within the packets in order to allow the succeeding nodes decode them. The vector of random coefficients of a node is updated for each of the nodes which perform the previous process of coding over randomized coefficients. These random coefficients enable a distributed transmission and a stronger robustness to losses and a redundancy can be added additionally to reduce packet loss. What is more, RLNC gives the intermediate nodes in the network the ability to re-encode (*recode*) without first decoding. This feature makes RLNC differentiate from classical encoding schemes such as Reed-Solomon, LT codes [86], LDPC... where intermediate nodes need to decode the received data before being able to recode it. The process of random coding is characterized by two main factors: (a) the generation size and (b) the field size.

(a) **The generation size** is the number of packets (symbols) that are coded together. It is shown that the size and composition of the generation has significant impact on the performance of network coding [97]. This value is modified based on the delay the transmission can have. For instance, in a video-call, the generation size is recommended to be small in order to reduce the delay, whereas in a a file download a bigger one is desirable (note that the packets can be decoded once the rank is full, hence the delay is larger if it has to wait until more packets arrive to decode them). Nevertheless, the computational complexity of RLNC using Gaussian elimination increases cubically with the generation size and therefore, making use of large generations is, in practice, less effective.

(b) **The Field size** of a Galois Field is the maximum amount of elements of the field that represent the coefficients used to code. Basically, the field size shows the range of the members of the field. A source packet p of size s can be binarily represented as an integer value within the range of the field. The use of a random code when the field size q is sufficiently large, will achieve the multicast capacity with a high probability. In practice, it is used a finite field of size $q = 2^m$ where m is the number of bits per field element. With this, the probability of not achieving a multicast capacity using a random linear network coding decreases exponentially with the number of bits per field element. Thus, to ensure a better independent linearity between coded packets, a larger field size must be chosen. Alternative stated, the larger the size is, the greater the probability to have linear independent coding packets is. However, the cost of having a big field size is seen in the efficiency inasmuch as, as the sixe of the field increases, the legth of the header also does because the coding coefficients becomes significant. The effect of the field size on the average decoding delay at the receiver is investigated in [98, 99]

This random feature of coding technique incorporates the property of ratelessness, that is, each message of size n has practically infinite coded packets. Furthermore, random linear network coding, in contrast to other traditional predefined coding, has the capacity to adapt to any transmission rate on the fly (Section 2.4.3). Because of these features and the ones mentioned before, RLNC is easy to implement and is considered as a suitable technique for dynamic topologies and varying connections.

In addition to these factors that characterize random linear network coding (generation size and field size), there is another element to consider which is the *coding method*, this is the different values and the position of the coding coefficients in the Coding matrix in order to reach certain goals in a transmission. There are different procedures to use random linear network coding that will be explained in Section 2.4. In the following sections, encoding, recoding and decoding methods will be explained.

2.3.1 Encoding

A node s observes some input stochastic processes $X_1, X_2, ...X_g$, each of them being a stream of bits that represent the source input information of the node. $Y_1, Y_2, ...Y_r$ denote the output bitstreams in node s transmitted to the next node n_1. Let $\underline{x}$ and $\underline{y}$ be the column vectors of the input and output processes observed in s. To encode this input information $\underline{x}$, a number of coefficients is needed, hence r elements from a Finite Field $\mathbb{F}_q$ of size $q = 2^m$ are chosen randomly to form the coding coefficient row vector $\underline{c}^\intercal = [c_1...c_r]$. Note that the node is generating more packets (r) than the ones it received (n), in order to add some redundancy. The process of encoding, then, will be performed as follows: a coded packet will be the result of the combination of the source packets with a coefficient vector. This can be represented as

$$y_1 = \begin{bmatrix} c_1 \cdots c_r \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \underline{c}_1^\intercal \cdot \underline{x}$$

The total of coded packets depends on the number of coefficient vectors that are used. The result of coded information can be represented as a matrix

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \underline{c}_1^\intercal \\ \vdots \\ \underline{c}_n^\intercal \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad \underline{y} = \underline{\mathbf{C}} \cdot \underline{x}$$

A packet is received by a node (recoder or decoder) and it is said to be an *innovative* packet to that coder if the encoding vector is not in the subspace ranged by the encoding vectors already received. In other words, a packet is *innovative* if its coefficient vector is linearly independent from the ones already received that corresponds to the same generation and with its reception it increases the receiver's knowledge space by one unit.

2.3.2 Recoding

The concept of recoding consists of a multiplication of the coefficients generated in the recoder to the coded packets and the coefficient vector. The coded packets represented by the column vector $\underline{y}$ contain the result coefficient matrix $\underline{\mathbf{C}}$, that was used to create these coded packets. These coefficients are extracted from the received packets. $\underline{\mathbf{G}}$ being the coefficients generated in the recoder, $\underline{\mathbf{M}}$ being the resulting coefficients of the previous coefficients and the new ones, the coded packets $\underline{z}$ are computed as

$$\underline{\mathbf{M}} = \underline{\mathbf{G}} \cdot \underline{\mathbf{C}}, \qquad \underline{z} = \underline{\mathbf{G}} \cdot \underline{y}.$$

2.3.3 Decoding

The decoding can be carried out once the decoder node has enough linearly independent coded packets. Once the decoder receives n *innovative* packets, it does not need to put more packets into the matrix because it can decode the n source symbols by Gauss-Jordan elimination. As the destination node t receives the coded packets $\underline{z}$, which can be represented by the linear matrix equation

$$\underline{z} = \underline{\mathbf{M}} \cdot \underline{x},$$

$\underline{\mathbf{M}}$ being the transfer matrix corresponding the result of all coefficients of all nodes, from the source node to the destination node. The decoding node places these received packets in a matrix. As the received coded data contains the result of all the coded coefficients of the transmission within the packets, the original information can be obtained by :

$$\hat{\underline{x}} = \underline{\mathbf{M}}^{-1} \cdot \underline{z}.$$

As mentioned before, the original data can not be obtained unless the received packets are linearly independent. Coefficient vectors must be also independent in order to be able to invert $\underline{\mathbf{M}}$, its determinant must not be zero.

2.3.4 Feedback in RLNC

Reliable communication can be achieved over point-to-point packet erasure channel using an Automatic Repeat reQuest (ARQ) scheme with feedbacks: when a packet gets *lost*, the sender re-sends it. Nevertheless, the throughput gets affected by the use of ARQ every time a packet is received. However, using network coding is necessary to achieve optimal throughput, even if acknowledgments are allowed. Wireless communications using forward error correction can not be enough to assure a reliable data transfer, hence, in order to achieve a good reliable point-to-point communication against packet losses with RLNC, automatic repeat request (ARQ) is used. Note that feedbacks are implemented in communications that need a very strong reliability and which packet loses need to be solvable. ARQ in RLNC works differently than the conventional ARQ because the receiving of a packet does not necessarily mean that the packet is *innovative* and useful. In conventional ARQ, when the sender receives an acknowledgments of a received packet it drops the acknowledged packets and transmits the next packet. However, in RLNC systems, another factors have to be taken into consideration as, instead of acknowledging packets, the receiver acknowledges the received *innovative* information (rank state of a coder). In other words, acknowledge mechanism confirms the degree of linearly independent rows in the coder matrix instead of the source packets.

2.4 Network Coding Codes

2.4.1 Fully-Coded RLNC

Originally, Random Linear Network Coding presented in [84] describes what it is called a fully-coded block RLNC. As previously mentioned, in practical implementations of RLNC, the data to be sent (N packets) is divided into groups of n symbols, called generations. Every generation has n source packets. An approach called Fully-coded RLNC, produces r coding packets (r being the result of the source packets n and the extra redundant packets) from the packets of the generation. Figure 2.4 represents the coefficient matrix $\underline{\mathbf{C}}$ used to code the source packets represented by a vector of packets $p_1, p_2, ...p_9$. In this example, 9 source packets are being coded with a coding ratio of 1.3 using the coefficient matrix $\underline{\mathbf{C}}$ with the coefficients taken from a binary finite field. As it can be seen all the row vectors of the matrix have random values of 0's and 1's. The coefficients of the received coded data are stored in a coding matrix initially empty and, as long as more data is obtained, this matrix gets filled until all the coefficients are located and the matrix is full.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boxed{p_1} \\ \boxed{p_2} \\ \boxed{p_3} \\ \boxed{p_4} \\ \boxed{p_5} \\ \boxed{p_6} \\ \boxed{p_7} \\ \boxed{p_8} \\ \boxed{p_9} \end{bmatrix}$$

Figure 2.4: Fully-coded RLNC coefficient matrix and packet representation

Notwithstanding the benefits of coding over routing in a randomized setting presented in [84], the method used has some drawbacks concerning the time a receiver has to wait until at least n coded symbols of a generation are received in order to start decoding, not to mention that these n symbols can not be recovered if they are not linearly independent, which happens when the field size used to generate the coefficients is too small. In other words, the per-packet delay is rather high taking into account that when the first packet reaches the node, it must wait until all the packets it needs arrive, to finally recover the first receieved one. Even if the generation size used is very small, this delay issue is not resolved. In addition to this drawback concerning the increase of latency obtained from the on-hold time of the node to receive enough packets, once the decoder has all n *innovative* packets, the complexity to recover all data performing Gauss-Jordan elimination is rather high.

To improve RLNC performance, encoding can be done *systematically* (Section 2.4.2) or *On the fly* (Section 2.4.3). The following *systematic Network Coding* approach, tries to reduce the computational complexity by sending source packets without encoding followed by redundant coded packets.

2.4.2 Systematic coding

Systematic network coding (Section 2.4.2) [89] [100], is a technique that consists on sending all the *innovative* packets and some coded redundant ones in order to reduce computational complexity which decreases the number of linearly dependent coded symbols and improves per-packet delay [101]. In [102], it is implied that a node should be able to instantly decode the received packets, within the meaning that when it obtains a packet, it can decode it instantly without the need of the reception of all the generation. By first sending source packets it allows the decoder obtain the original information when the packet reach the decoder and, if one or more than one packet is lost, it waits until the coded packets arrive (located at the end of the generation) to recover the lost packets. As shown in Figure 2.5, the coefficient matrix $\underline{\mathbf{C}}$ is divided into two sub-matrices where the upper part is the identity matrix (source packets) and in the lowest part the random coefficients to generate coded packets. In comparison to fully-coded RLNC, by using Systematic RLNC a better delay is achieved [103].

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 2.5: Coefficient matrix using systematic RLNC (binary field)

2.4.3 Online Network Coding (on the fly)

On the fly coding, also known as online NC [104], enables encode as original data is received or as the rank is increased. The node can generate coded packets with the current information available, it does not require full rank matrix [56] [52]. This approach is not influenced by buffering issues of block codes as the encoder does not need to store all the information to generate packets. Online NC can be performed with the full coded coding or the systematic coding.

Chapter 3

Introduction to PACEMG protocol

3.1 NCKernel library

In this project, a protocol called PACEMG is presented and is finished by developing the recoder part. This protocol uses what a library called Network Coding Kernel (NCKernel) developed by the Deutsche Telekom Chair of Communication Networks. This library is used by other protocols apart from this one. NCKernel uses a library called Kodo from Steinwurf, to perform all the coding part in the protocol.

3.1.1 Libraries of NCKernel

Fifi

In order to have a fast RLNC it is a requirement to use a library called Fifi. Fifi is a low level library use for handling finite fields arithmetics, including $GF(2), GF(2^4), GF(2^8)$ and $GF(2^{16})$. To a better deployment, these optimizations are compiled into single binary which lets the user not think about the hardware the code is being run on. It is written in c++ but there are language bindings in java and python.

Kodo

Kodo [105] is a library written in c++ developed by Steinwurf that implements erasure coders. This erasure coding library puts focus on network coding algorithms and codecs which makes it perfect for network coding protocol creation. Through a simple Application Programming Interface (API) the user can employ network coding functionality. There are multiple git repositories that correspond to each codec of Kodo (e.g. kodo-rlnc, kodo-fulcrum, kodo-reed-solomon, kodo-core...). The language binding the protocols presented in Section 3.2 will be using is kodo-c, even though it is a deprecated project. Kodo-c library was created to provide a c wrapper but it uses earlier implementation of RLNC codes, with this, the newest version (not deprecated) of Kodo-c is kodo-rlnc-c. To avoid delaying the delivery of decoded symbols more than absolutely necessary, Kodo supports partial decoding of blocks, which means that in order to perform partial decoding, the coder uses the rank and tracks when the coefficient matrix resolves to pivot symbols.

3.2 Protocols

3.2.1 PACE

A protocol proposed to face some telecommunications issues in the Deutsche Telekom 5G chair in TU-Dresden is called PACE [1]. This approach tries to solve the latency problem when all coded packets have to be received before start decoding them. This issue that the systematic and non-systematic network coding encounter, can be fixed with the positioning of coded symbols among the systematic ones in a generation. As a result, the mean delay for the recovery of the source packets in the destination node is reduced by using PACE. The main idea is, instead of sending all coded packets or sending first the systematic packets and the FEC packets at the end, the nodes will send the source packets and in between, coded packets will be created from the previous source ones. In other words, a generation is divided into n/ϵ sub-generations (n and ϵ being the number of symbols in a generation and the number of FEC packets in a generation respectively) with the purpose of placing at the end of the each sub-generation the coded packets generated from the source packets of the current sub-generation and the previous ones. To understand this in a better way, first I need to introduce the following concepts.

Field size:	The Galois Field size $GF(2^n)$ which over all the elements of it the coefficients are chosen independently and uniformly.
Generation:	A group of n source packets.
Symbols (Gen. size n):	The maximum number of source packets in a generation n that can be linearly combined to form a transmission symbol.
Coding Ratio (c):	The percentage of packets that will be generated from a generation (e.g. If a generation has 10 symbols and the coding ratio is 130%, there will be 13 packets in that generation to transmit).
Symbols Size:	The Size of a Packet in the generation (in Bytes)
FEC packets (ϵ):	The number of coded packets (forward error correction) in a generation $\epsilon = \lceil (c-1)n \rceil$
Tail packets:	The Coded packets that will be placed apart from the "coded ones" at the end of the generation to add redundancy and robustness.

The best way to explain how this protocol works is by doing a comparison between the systematic network coding, the traditional NC and PACE . As noted in Section 2.3, in order to create the transmission packets $\underline{y}$ it is essential to have the coefficient matrix $\underline{\mathbf{C}}$ and the source symbols $\underline{x}$ ($\underline{y} = \underline{\mathbf{C}} \cdot \underline{x}$) but, as the source symbols are unchangeable, what makes one code different than the other is, at the end, the design of the coefficient matrix $\underline{\mathbf{C}}$. Let's say there is a generation of $G = 9$ source packets and a coding ratio of $C = 130\%$, then, the encoder would be creating $\epsilon = 3$ coded packets (FEC packets) which at the end, the number of total transmitted packets would be 12 . If systematic RLNC is performed, the FEC packets would be placed at the end of the generation and the source packets would be placed at the beginning of the generation (Figure 2.5). On the other hand, applying PACE, the coefficient matrix would look slightly different (Figure 3.1). The generation is divided in n/ϵ sub-generations each of them with n_{sub} source symbols and some coded symbols at the end of each sub-generation. The coded symbols are generated from the source symbols of the correspondent sub-generation and the previous ones of the generation (Dark color in Figure 3.1).

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 3.1: PACE coefficient matrix (binary field) of 9 symbols and a coding ratio of 1.3

Every sub-generation is surrounded by a blue rectangle. At the end of each sub-generation there is a darker color row which corresponds to the coefficients of the coded symbols.

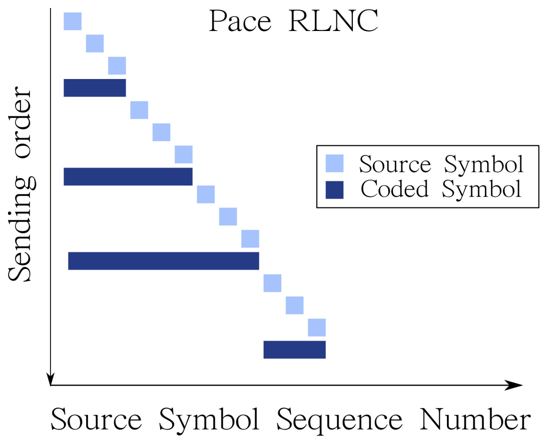


Figure 3.2: PACE coding time representation with Containers of 9 symbols and a coding ratio of 1.3

The squares are systematic symbol and after some source symbols are sent, one coded combining the ones before is generated. The first three coded packets pertain to a generation, consecutively, the three last systematic and coded symbols belong to another generaion.

Packet Structure

The packet structure of PACE protocol is shown in Figure 3.4. The header used tells the forwarding node, which generation the packet corresponds and the rank that the sender has when it sends the packet. When the receiver receives the packet, sends a feedback packet with the rank of the coder and the generation number. With this information the sender knows what is the rank number.

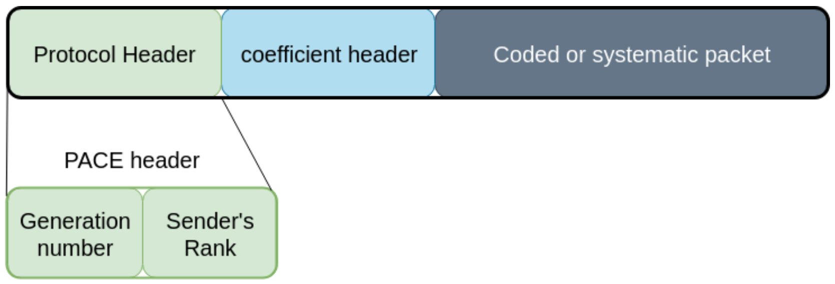


Figure 3.4: Packet structure of PACE protocol

Despite the fact that the source packets delivery mean delay is reduced respect the systematically RLNC, the protocol confronts some issues in the matter of arrival delay. PACE protocol can deal with G symbols, however, it does not work as it would be wanted in the following situation. Imagine there is a scenario with one encoder (source node) and one decoder (destination node) and, the $Node_{src}$ wants to send more than 20 source packets to the destination node $Node_{dst}$. Using the same values as before ($G = 9$, $C = 130\%$, $\epsilon = 3$ and a total packets generated = 12) the encoder would be building at least 3 generations sequentially and would be sending 12 packets to the decoder. The problem is presented in Figure 3.5: when the encoder sends more packets and, due to the use of multi-path communication, the packets go through different nodes using various paths and reach the decoder with some messiness, newer generations arrive before packets from older generations causing the lost of the oldest packets.

When the decoder receives a packet from generation 2 before a packet of generation 1 and the decoder needs more packets in order to fully decode all generation 1, then it has to *flush* that generation and delete it to start the new one (generation 2). The container *flushes* the generation in the sense that it decodes all the information that is uncoded in the matrix and puts it in the uncoded queue, but it can not recover the information that is not. Consequently, the decoder can only recover part of the information of the deleted generation.

As it has been seen in 3.2.1 to a certain extent, PACE improves the delay when decoding packets, however, it wouldn't be convenient using in multi-path communications where it is very likely to receive disorganized information. This problem is practically solved in PACE multi.generation protocol (PACEMG) in Section 3.2.2.

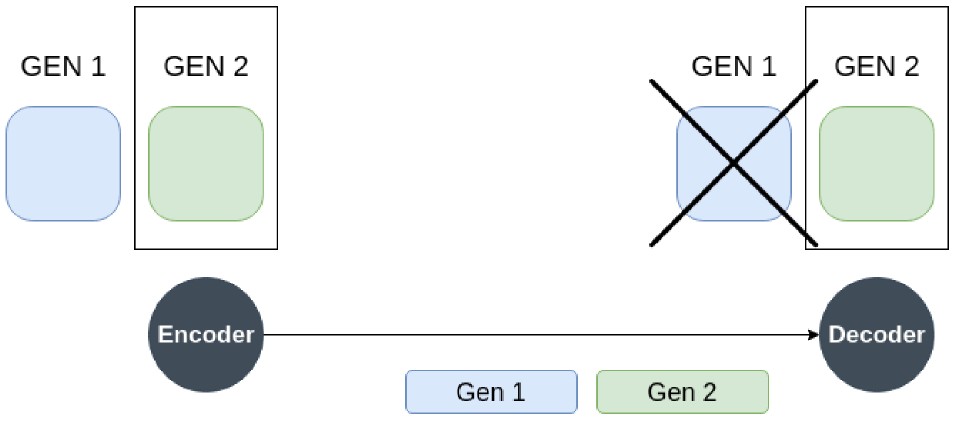


Figure 3.5: PACE used in a scenario with jitter and losses.

The encoder has sent all the packets from generation 1 and starts sending the ones from generation 2, but because of the jitter, the last packet from generation 1 arrives after the first packet of generation 2. The decoder, when the packet from generation 2 arrives, deletes the generation 1.

3.2.2 PACEMG

As aforementioned (Section 3.2.1), notwithstanding PACE has improved the per-packet delay of the packets when losses occur, the disorder of the arrival of the packets, when they belong to different generations, induces to information getting lost. PACEMG seeks to solve this constraint by, instead of managing only one generation in the coders, handling a limit number of generations simultaneously. The coders can have more than one generation at the same time, thereby ensuring it will not be necessary to delete the generation when a symbol of the next generation is received, provided that the number of generations don't exceed the limit established in the coder. The fundamental characteristic of this protocol is the capacity to administer more than one generation. In each coder a generation is managed by a *container*. Imagine the following situation: a coder (encoder, recoder or decoder) receives the first packet of $Generation_1$, the coder creates a new *Container* corresponding to that Generation. A *Container* can be thought as a sub-coder that administers all the information that corresponds to a specific generation. When the coder receives afterwards a packet of $Generation_2$ it creates another *Container* with the corresponding generation if it does not have one container with that Generation already. Every time a packet of $Generation_1$ or $Generation_2$ is received, the coder has to call the $Container_1$ or $Container_2$ to handle that packet.

In the following sections both, the packet structure and the feedback packet structure will be presented together with their functionality and use.

Packet structure

The packet structure of the PACEMG protocol is shown in Figure 3.7. The packet contains two headers, on one hand there is the protocol (PACEMG) header that is generated by the protocol and contains some information regarding the coder state and, on the other hand, the coefficient header that informs the coder the encoding procedure. It contains some information regarding the coding state and some other information that is necessary in order perform a reliable communication using feedback.

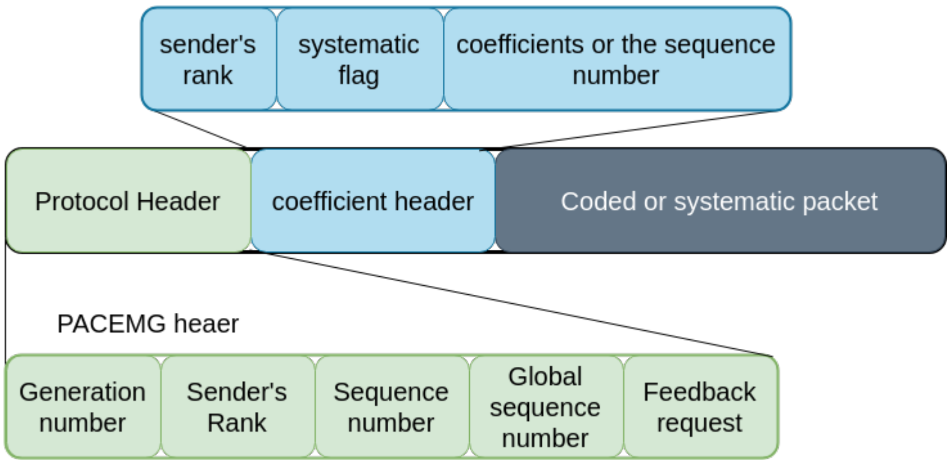


Figure 3.7: PACEMG coded packet structure

As it can be seen in the Figure, the PACEMG header contains:

- Generation number (4 bytes)
- Sender's rank (2 bytes): the rank of the container that is creating the packet when it codes it.
- Sequence number (2 bytes): The sequence number of the generation. It is the number of sent packets of that container.
- Global sequence number (4 bytes): The number of sent packets of that Coder.
- Feedback Request Flag (1 byte): If the flag is set to '1', then it is demanding the receiver to acknowledge the packets with a feedback packet (Figure 3.8).

The protocol header is the header of PACEMG needed for the purpose of a good management of the containers of every coder. It is also required with feedbacks, because the sender, and also the receiver, need to control all the information for a good communication. With the generation number, the coder is able to know which container needs to acquire the input packet. Once the the container has the packet, it saves the sender's rank, sequence number and global sequence number in order to let the sender know what was its rank when the receiver obtained the packet. It also needs to know if the sender demands feedback or not. The sender's rank and the sequence number is needed for the feedback, but the received does not use these two values.

Feedback in PACEMG

The feedback packet (Figure 3.8) is composed by two main parts: from one hand, the "darker color" (first bytes of the packet) gives the sender information about the oldest generation it currently has operative in the coder in order to let the sender delete older information, and, on the other hand, the "light color" (the rest of the bytes of the packet) is the set of feedback packets of every current container (ordered sequentially from the oldest to the newest container).

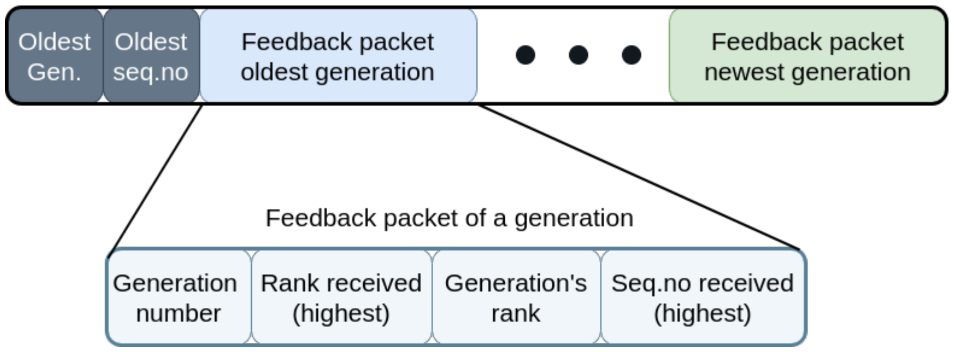

Oldest Gen.
Oldest seq.no
Feedback packet oldest generation
Feedback packet newest generation
Feedback packet of a generation
Generation number
Rank received (highest)
Generation's rank
Seq.no received (highest)

Figure 3.8: Feedback packet of PACEMG protocol

In every Feedback's container packet there is :

- Generation number (4 bytes)
- Highest sender's rank (2 bytes): the rank of the received packet (is the rank of the sender when it sent it). This value is updated every time a packet is received with a higher rank, if the packet received has a lower rank, it is not updated.
- Generation's rank (2 bytes): The rank of the container at the moment of sending the feedback packet.
- Sequence number received (4 bytes): The sequence number of the received packet.

It is taken from the header of the packet and is updated every time a packet with a higher sequence number is processed.

It should be pointed out that the size of the feedback packet is variable as it can be either 3 containers in the feedback packet or 7. To explain in a better way how this process works, it is needed the use of a graphical representation (Figure 3.9 and Figure 3.11). This two figures represent a communication scenario with one encoder and one decoder. The encoder has sent packets from three different generations and it has not deleted any of them. The decoder has one deleted generation (GEN 1) because it has already decoded all the packets from that generation, and apart from this, two generations currently available to receive packets. Figure 3.9 illustrates the moment when the encoder is sending a packet from *generation*$_2$ (green packet) and, at the same time, the decoder is sending a feedback packet. The Feedback packet contains information regarding the oldest container generation currently operative (first bytes) followed by the feedback packets of *generation*$_2$ and *generation*$_3$ (green and yellow respectively).

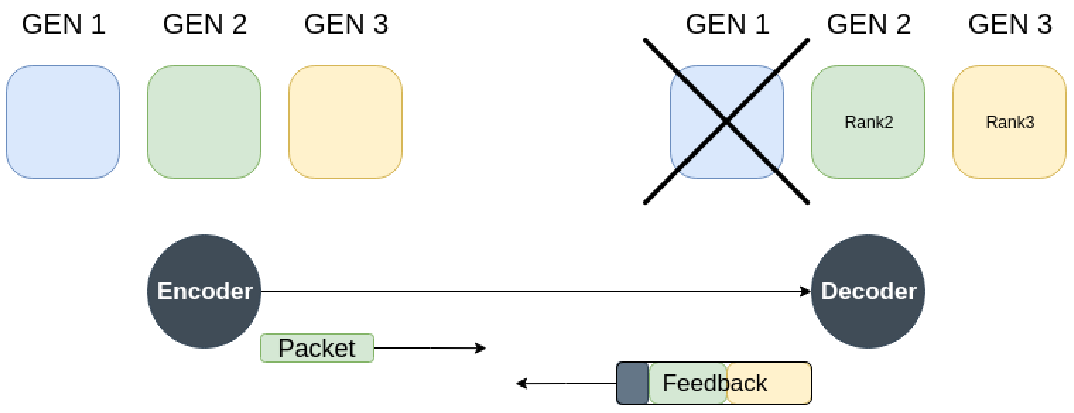


Figure 3.9: Representation of a scenario using PACEMG with feedback I

Encoder sends a packet from of the *generation*$_2$ (green color) and the decoder sends the feedback in that moment to the 3ncoder

When these two packets arrive to each destination (the feedback packet reaches the encoder and the coded packet reaches the decoder) they are processes by each coder.

On one hand, assume that the received packet (green) in the decoder was the last packet $generation_2$ needed for recovering all the information, then the decoder would delete $Container_2$. On the other node, there is the feedback packet, which tells the encoder what is the oldest currently working container of the decoder and, therefore, it can delete all the older containers because the decoder will not need more information from that container. Later, the encoder extracts the feedback information from $generation_2$ and $generation_3$. The encoder then, checks the rank of $generation_2$ and sees that the rank of the decoder when it sent the feedback was almost the number of symbols and, with the previously sent coded packet from $generation_2$ (if received successfully), the decoder will have full rank and the encoder deletes that container too. It is assumed that some previously packets from $generation_3$ have been lost and the decoder will not be able to decode all the information. The encoder, when the feedback packet of $generation_3$ is checked, extracts the last sequence number received, the last rank received and the current rank. With this information, it first checks what is the rank difference between the encoder's rank (when it sent the packet) and the decoder's rank and, apart from this, sees what was the newest packet it received (last sequence number received). Here, the encoder, takes into consideration that all the previous packets have been received or lost and it sees how many packets are to be sent. Finally it computes, in case the packets to be sent are successfully received, how many packets the decoder will need additionally and whether the decoder will be able to have full rank or not. Figure 3.11 illustrates the case where the encoder needs to send more packets because the decoder will not have enough to recover the data.

Note that every feedback is from point-to-point communication. This is to say that if there is a $encoder - recoder - decoder$ every node manages the feedbacks as a sender and as a receiver. The encoder would send a packet to recoder and the recoder would send acknowledgments to the encoder, the recoder then would send coded packets to the decoder and the decoder would send acknowledgments to the recoder, but it would not forward the acknowledgments to the encoder.
It is clear that in order to implement ARQ mechanism, the nodes need to retain the information so as to send more information if needed. Since the encoders and recoders

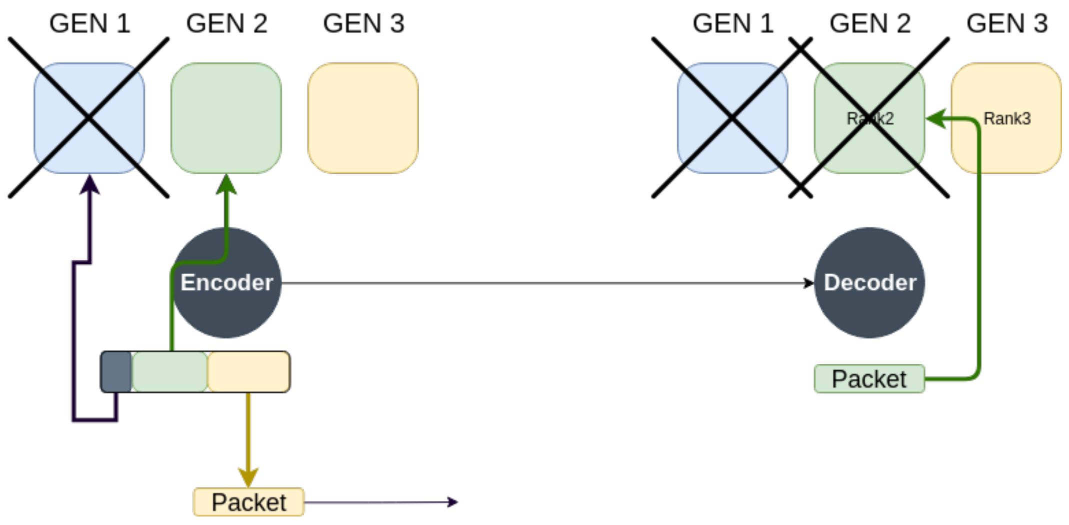


Figure 3.11: Representation of scenario using PACEMG with feedback II.

The encoder extracts the information from the feedback packet and, knowing how many packets has that container sent after the sequence number that is in the feedback packet, it computes the amount of packets it should sent if it considers there won't be enough linear symbols in the decoder to have full rank (of that generation).

can not delete a container unless a feedback packet is received with the information that the receiver node has deleted that generation, when ARQ is not used, the oldest container is deleted every time it reaches the maximum number of containers. However, if the number of containers is very high, the stored information (all the maximum containers) gets dense. The reason why feedback request is done by the sender is that it can delete information it thinks is not needed any more, or at the end of a sent generation. So, when the whole generation is sent, the sender could send a feedback request with the last packet of the generation and then if the receiver obtained it it would send a feedback packet and the encoder would be able to delete the generation or the previous ones. The rate of the feedbacks request can vary depending on the need of the reliability: as long as feedbacks is used, the recovery of lost packets is managed, and in the worst case where the sender has already deleted a generation, the information is not recovered. Moreover, the use of feedbacks can also help to recognize the amount of lost information that is occurring, which can be useful to adapt to such loss rate by increasing the coding ratio or increasing the feedbacks requests.

3.3 How NCKernel and PACEMG works

In order to understand the functionality of RLNC together with kodo and nc_kernel, a typical scenario with an encoder-recoder-decoder will be explained. The coding, recoding and decoding process works as follows: the node that is sending the data divides it into packets of size s bytes, this data then is introduced into the encoder with the following function:

nck_put_source(encoder, packet)

What this function does is to basically call the encoder and give the encoder the source packet address, the encoder generates a new container if it does not have any, or the one that has is full, and introduces the source packet into the coder of that container. The generation, later, updates its rank and checks how many packets it needs to send with this rank. To check if there are coded packets to send (or systematic packets), the node then calls:

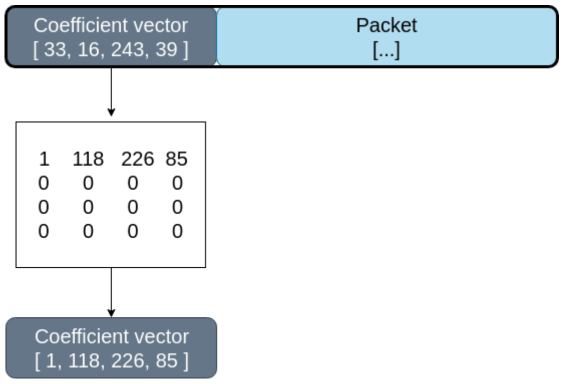
nck_has_coded(encoder)

This returns $true$ or $false$ if it has coded packets to send or does not respectively. If it has coded packets then it creates a new empty packet of size (coder *coded_size*) in order to let the encoder put the new coded packet in this packet by doing

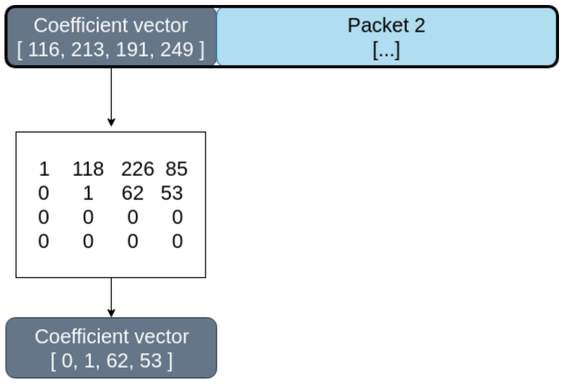
nck_get_coded(encoder, packet)

When this function is called, the encoder gets the coded packet from the coder of that generation and introduces the protocol header (Figure 3.7). The information of the header is obtained from the container where the coded packet is extracted. Then, the node obtains a new systematic or coded packet which sends to the forwarding node. Suppose this forwarding node is a recoder. The recoder node acquires this coded packet and it puts it into the recoder by *nck_put_coded(recoder, packet)*

The recoder extracts the header information and gets the generation number and calls the container with that generation number, if it does not have any container with the generation it received, it creates a new one with the characteristics of the recoder. With the header information of the packet, the container saves the rank number and the sequence number of the received packet and checks if the sender is asking for a feedback packet. Finally puts the coded packet inside the coder of the container. By introducing a coded or systematic packet into a coder, the coder pulls out the coefficient vector contained in the packet, appends it to the coefficient matrix and performs partial Gaussian elimination. This process ensures not only that the size of matrix $\underline{\mathbf{C}}$ is limited to nxn (n is the number of source symbols per generation), but also shapes $\underline{\mathbf{C}}$ into a lower triangular matrix. This process of reshaping the matrix is shown in Figure 3.13a and Figure 3.13b. The diagonal of $\underline{\mathbf{C}}$ has 1's and 0's. The values that contain a 1 in the diagonal has 0's on the left and values on the right. If the input packet is a systematic one, it only contains a 1 in the corresponding packet and the rest of the row are 0's.



(a) The first packet is introduced into the coder



(b) The second packet is introduced to the coder

Figure 3.13: Representation when symbols are introduced in the Kodo coder

When this is done, the rank of the coder is updated and it increases the packets it needs to send. It only increases this *to_send* value if the rank is increased of the coder generation respect the the rank the generation had before introducing the packet. The

node then to see if there are coded packets that need to be sent, it needs to call the *nck_has_coded(recoder)* that is called in the encoder (but calling the recoder coder instead of the encoder). The coder of the container where the coded packet needs to be taken from, generates a new vector of coefficients and creates the new packet with the resulting coefficient vector.
The decoder, when it receives information (*nck_put_coded*), it does the same process as the recoder has done. To get the source data from the decoder:

nck_has_source(decoder) *nck_get_source(decoder, packet)*

This process of inputting the coefficients and the coded or systematic packets received into the coder matrix is called *pivoting the decoding matrix*. This procedure is done by every recoder and decoder. *Pivoting* is the operation of selecting of a sequence of elements from a matrix (non-zero element) and re-ordering the matrix by interchanging rows and columns in order to bring the pivot to a fixed position. By putting a value in the coder matrix, the coder performs *partial pivoting*, this process consists of selecting the entry with largest absolute value from the column of the matrix. The objective of pivoting is to make an element above or below a leading one into a zero. When the number of pivots equals the number of symbols of a generation, the matrix is reordered according to the sequence of selected pivots by exchanging rows and columns such that the *i-th* selected pivot is now the *i-th* diagonal element of the new matrix. Note that the rows of the right-side matrix of the linear system of equations need to be reordered accordingly such that the solution to the linear system remains unchanged. All this process is done by the Fifi and Kodo library.

Chapter 4

Implementation

The work of this thesis centers on the development and improvement of the procol PACEMG presented in Section 3.2.2. As explained before, this RLNC protocol uses a library called kodo-c, however, it is not supported anymore. Facing an issue provoked by the use of this library, four versions of the recoder of PACEMG have been done. In this chapter these four versions are explained: a recoder that does not performs systematic coding using kodo-c and using kodo-rlnc-c library and a recoder that performs systematic coding using kodo-c and kodo-rlnc-c library.
Beforehand, a simple change done in the decoder to fix a behaviour when disordered packets appear, is explained. The modification done in the decoder (which is also implemented in the recoder) for the correct performance of the protocol with non-sequential packet arrival is done in the part of the coder where it inputs the received packets. Whenever a packet is received, the rank number of the sender (encoder or recoder) and the sequence number of the packet that are located in the packet header are stored in the container. However, if a packet ($packet_1$) arrives after another packet ($packet_2$) with lower sequence number and lower rank than $packet_1$, the number of the sequence number and the rank of the sender saved in the container of the receiver can not be the ones from the last packet received ($packet_1$). The reason for this, is that when the receiver generates the feedback packet, it needs to send the last rank and sequence number of the sender received for the sender to compute the number of packets it needs

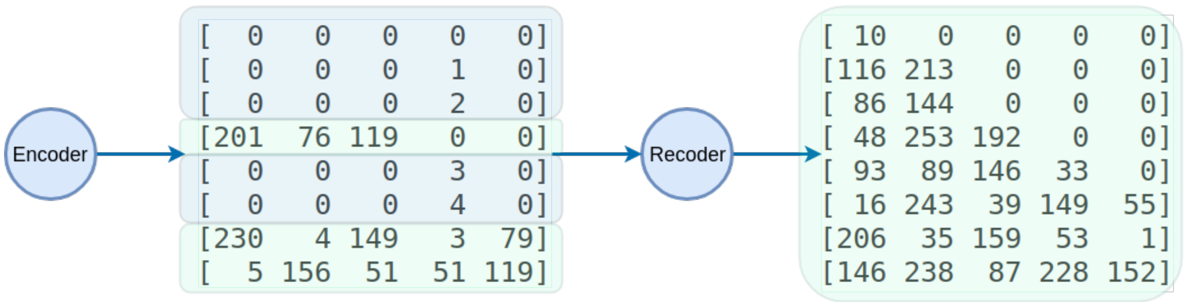
to send, and, if the sequence number and rank of $packet_1$ (the last received) are placed in the feedback packet, the sender will think that the receiver has only received $packet_1$.

4.1 PACEMG recoders:

The main objective of this thesis is to improve the PACEMG protocol by producing the recoder, which has the both functions, to encode and decode. It recodes the packets that are received, it decodes them too and manages feedbacks. Two versions of the recoder have been developed: the first version is a recoder that does not perform systematic coding and the second version is the systematic version of the recoder. These two versions will be explain in the following points.

Version 1 **of PACEMG Recoder: non-systematic**

As indicated, PACEMG protocol uses the Kodo-c library (a deprecated library) in order to use the functionalities of Kodo_core and Kodo_rlnc libraries. This library uses and treats the recoder as a decoder, which means that the decoder can recode. Nevertheless, it can not recode systematically, but on the fly. In On-the-fly RLNC, symbols can be encoded as they are available in the coder and the coded data leaves the recoder as decoding progresses. This is, every symbol received, the recoder combines it with the previous ones in the generation (only if the rank is increased). This is different from traditional block codes where all data has to be available before encoding or decoding takes place. The recoder, can not send a systematic packet without coding if it is put in the coder and later taken out (*nck_put_coded*, *nck_get_coded*). Therefore, this protocol can only perform systematic coding if it is a encoder, which is not very useful if the aim of this protocol is to receive uncoded packets in order to be able to obtain data once the first information is received. Yet, as the library has this functionality, the first recoder developed does not perform systematic recoding, but on-the-fy. Figure 4.1 shows an example of the generated coefficients of the protocol when they leave the encoder and the coefficients generated by the recoder. It can be easily viewed that the recoder performs *on the fly* fully coding, as the first element of the matrix only combines the first packet, the second combines both, and so on.



```
[  0   0   0   0   0]
[  0   0   0   1   0]
[  0   0   0   2   0]
[201  76 119   0   0]
[  0   0   0   3   0]
[  0   0   0   4   0]
[230   4 149   3  79]
[  5 156  51  51 119]
```

```
[ 10   0   0   0   0]
[116 213   0   0   0]
[ 86 144   0   0   0]
[ 48 253 192   0   0]
[ 93  89 146  33   0]
[ 16 243  39 149  55]
[206  35 159  53   1]
[146 238  87 228 152]
```

Figure 4.1: Coefficient matrices generated in the encoder and the recoder

The encoder sends the packets with the coefficients of the matrix on the left. The first three packets are systematic, the next one is a coded packet combining the three previous ones, then two more systematic packets and the last two ones are coded packets generated from all the previous ones. The coded packets that the recoder creates, are the result of the combination of the source packets employinh the coefficients of the matrix situated on the right. None of the packets that leave the recoder are systematic.

A simplified representation of events when some data is received by the recoder is shown in Figure 4.3.

Feedback

The feedback approach in the recoder changes regarding the encoder. The encoder, as it does not decode the information, does not need the conatiners when it knows that the next node has full rank. Consequently, the encoder, whenever it receives a feedback packet, it deletes all the old information the next node has previously aknowledged.

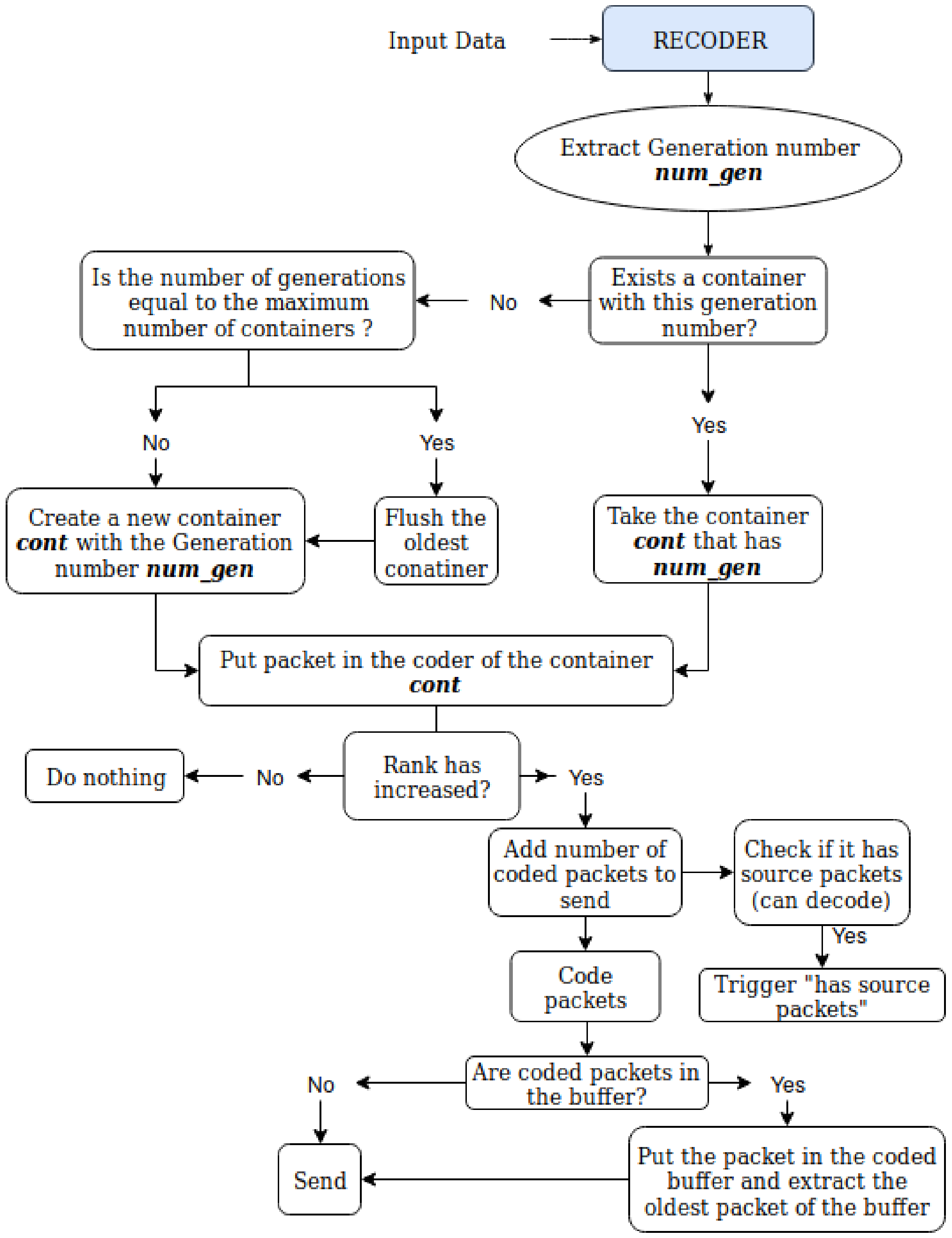

Input Data
RECODER
Extract Generation number num_gen
Exists a container with this generation number?
No
Is the number of generations equal to the maximum number of containers ?
No
Yes
Yes
Create a new container cont with the Generation number num_gen
Flush the oldest conatiner
Take the container cont that has num_gen
Put packet in the coder of the container cont
Do nothing
No
Rank has increased?
Yes
Add number of coded packets to send
Check if it has source packets (can decode)
Yes
Code packets
Trigger "has source packets"
No
Are coded packets in the buffer?
Yes
Send
Put the packet in the coded buffer and extract the oldest packet of the buffer

Figure 4.3: Process when a packet is received in the recoder

However, if this process is done in the recoder and the recoder also wants to decode the data, there will be a loss of information. When decoding in the decoder, firstly the decoder checks if the oldest container has data to decode and, if it has, it decodes and deletes the container (if the decoding has been fully and succesfully performed). Note that, if 3 containers are available at that moment, the decoder only checks the last container. However, in the event where the oldest container (suppose it is generation g) does not have full rank, the decoder waits until the maximum number of active containers is reached in roder to flush that container and recover the maximum amount of information possible to later on, delete it. Then, next time the node checks if it has information to decode, the decoder will have as a oldest container another generation $g + 1$. In this exact case, the decoder sends a feedback packet to a recoder (with the oldest generation number as $g + 2$ because suppose that when it sends the feedback packet the generation $g + 1$ is also already decoded) and the recoder does not have full rank in generation g, which means it can neither decode that generation nor the following ones. If the recoder behaved as the encoder does, it would automatically delete all the containers older than the acknowledged, g and $g + 1$, which, if $g + 1$ had full rank, the data would not be decoded and, moreover, if some data could be recovered from container g, this data would be also lost. With the exmaple above, the recoder loses information that needs to decode.
A solution to fix the performance above is: when a feedback packet is received, instead of automatically delete the older containers, it checks if the packets from that container have already been decoded which, in that case it would delete it. In case that the container is not decoded, the container would be flushed and if any information could be recovered, the container would be deleted. Only that container would be deleted. In addition to checking the number of the oldest generation, the recoder also checks the rank of the decoder and whether it will need more information. If not information is needed from a certain generation, the recoder checks if that generation is already been decoded and if it is, the container is deleted. By doing this, the recoder is able to consecutively decode the generations in the case when some generation can not be decoded. However, in some implementations of the recoders, the purpose is not to

re-code and decode but to re-code. In that case, the recoder could implement the same feedback method as the encoder in order to save memory. Clarify that even when the recoder can not decode at a particular moment the generations that are not the oldest one, it is still able to send coded information of that generation, hence, allowing the decoder have the same rank as the recoder.

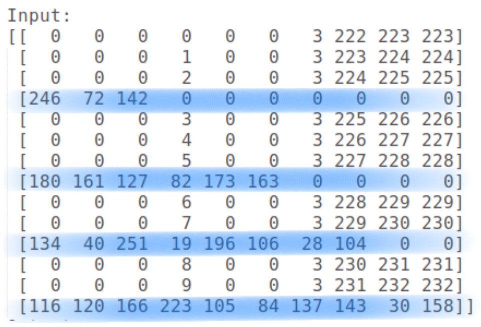
The use of a deprecated library faces some issues. Kodo_rlnc library has some new functionalities that can be used in Kodo-rlnc-c but not in kodo-c. The most important one (thought in this thesis) which provokes the incorrect performance of the protocol, is that the coefficients that every recoder generates in each generation are always the same. These coefficients are random in the sense that are generated randomly using a seed. Nonetheless, the seed used by every coder (container) in every recoder is the same, inducing to the production of the same sequence of coefficients in each of are equal. Even though the encoder does generate different coefficients for every new generation and in every different simulation, the recoders do not.

This version of the protocol faces an issue: performing fully coding is not convenient when the aim of the protocol is not only to be robust, but also to be a short delay RLNC protocol. If the recoders performs coding with every *innovative* packet that arrives, at the end, if the packets arrive non-sequentially, the decoder will have to wait until all the packets from that generation arrive, which is why this protocol uses *systematic* coding, to avoid this unnecessary delay.

Version 2 of PACEMG Recoder: systematic

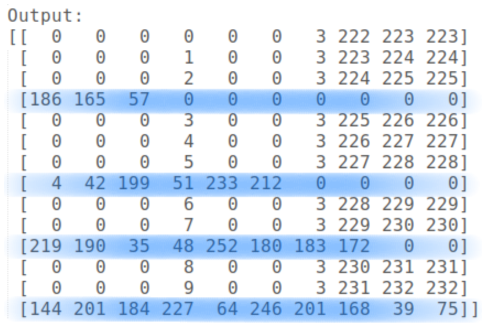
The second version of the recoder the PACEMG protocol consists on using *systematic on-the-fly* recoding. Taking into account that the Kodo coder can not send systematic packets, the way it has been done in order to do so does not let the Kodo coder code the packet. The procedure it employs is exactly the same as the recoder from *version 1* except when the coded packet enters the recoder that the process is the following: A recoder receives a packet (can be systematic or coded) and puts it in the recoder by $nck\_put\_coded(recoder, packet)$. When it enters the recoder, after extracting the generation number and the rest of the information located in the PACEMG header, it checks if the systematic flag is on or off. This systematic flag is placed in the coefficient header (the header of the kodo), concretely in the 4th byte of the header. Therefore, the header does not need to be extracted because by only checking that position in the packet data buffer, the recoder is able to know. If this the systematic flag is off, the recoder does completely the same as the recoder $version1$, however, if that packet is a systematic one, the procedure changes. Another null packet ($auxiliar$) is created with the same size as the inputted one. All the data from the introduced packet is copied into the new *auxiliar* packet, subsequently, the search for the corresponding container is done. Once the matching container is taken, the received packet is placed into the Kodo coder of the generation and the rank increase is verified. As the recoder does not send information unless the rank is increased, if it is, the recoder, unlike the recoder *version1* that only updates the *to_send* variable, it places the copied systematic packet ($auxiliar$) in the *coded_queue* of the container with the correspondent PACEMG header. Note that this *auxiliar* packet is employed because the Kodo coder manages the packets and makes changes in the inputted packets so, if the received packet, that was already put in the Kodo coder, was also put in the *coded_queue*, the packet would not be the same as the received one due to the changes the Kodo coder would have made. When the recoder, after all this process is done, wants to take out coded packets by $nck\_get\_coded(recoder, packet)$, firstly, it takes out the coded packets from the *coded_queue* and if, subsequently, the queue is empty and more packets are needed to be sent, it takes coded packets from the kodo coder. Since the systematic packet was inputted in the *coded_queue* it will be sent without coding and it will be systematic packet even though recoders in Kodo do not support systematic recoding.

An example of tge coefficients generated in a recoder can be seen Figure 4.4. The coefficients of the packets received and generated by $node_1$ (the second recoder) are the same (without appling jitter). However, the recoder also sends systematic packets (identical as the ones it received). The systematic packets are sent without changing anything, but the coded ones are taken from the coder of the generation (and the coded coefficients are the same).



```
Input:
[[  0   0   0   0   0   0   3 222 223 223]
 [  0   0   0   1   0   0   3 223 224 224]
 [  0   0   0   2   0   0   3 224 225 225]
 [246  72 142   0   0   0   0   0   0   0]
 [  0   0   0   3   0   0   3 225 226 226]
 [  0   0   0   4   0   0   3 226 227 227]
 [  0   0   0   5   0   0   3 227 228 228]
 [180 161 127  82 173 163   0   0   0   0]
 [  0   0   0   6   0   0   3 228 229 229]
 [  0   0   0   7   0   0   3 229 230 230]
 [134  40 251  19 196 106  28 104   0   0]
 [  0   0   0   8   0   0   3 230 231 231]
 [  0   0   0   9   0   0   3 231 232 232]
 [116 120 166 223 105  84 137 143  30 158]]
```

The input come from the encoder and the packets are systematic. The colored rows are the coefficients of the coded packets, the other ones are the systematic packets.



```
Output:
[[  0   0   0   0   0   0   3 222 223 223]
 [  0   0   0   1   0   0   3 223 224 224]
 [  0   0   0   2   0   0   3 224 225 225]
 [186 165  57   0   0   0   0   0   0   0]
 [  0   0   0   3   0   0   3 225 226 226]
 [  0   0   0   4   0   0   3 226 227 227]
 [  0   0   0   5   0   0   3 227 228 228]
 [  4  42 199  51 233 212   0   0   0   0]
 [  0   0   0   6   0   0   3 228 229 229]
 [  0   0   0   7   0   0   3 229 230 230]
 [219 190  35  48 252 180 183 172   0   0]
 [  0   0   0   8   0   0   3 230 231 231]
 [  0   0   0   9   0   0   3 231 232 232]
 [144 201 184 227  64 246 201 168  39  75]]
```

As the received coded packets (painted in blue) come from the encoder, the sent coded packets will have different coefficients. Nevertheless, the systematic packets are exacly the same as the ones received.

Figure 4.4: Version 2 recoder. Coefficients received and generated by $node_1$ (first recoder) with a coding ratio of 140.

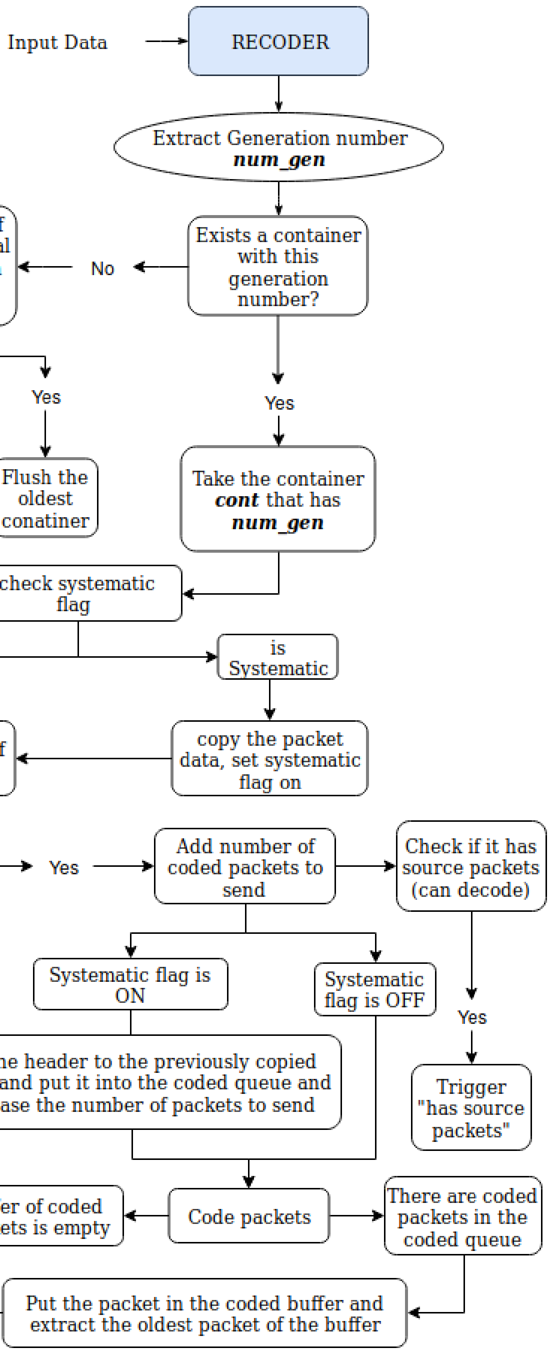

Input Data
RECODER
Extract Generation number **num_gen**
Exists a container with this generation number?
No
Is the number of generations equal to the maximum number of containers ?
No
Yes
Yes
Create a new container **cont** with the Generation number **num_gen**
Flush the oldest conatiner
Take the container **cont** that has **num_gen**
check systematic flag
Is not a systematic packet
is Systematic
Put packet in the coder of the container **cont**
copy the packet data, set systematic flag on
Rank has increased?
Yes
Add number of coded packets to send
Check if it has source packets (can decode)
No
Do nothing
Systematic flag is ON
Systematic flag is OFF
Yes
add the header to the previously copied packet and put it into the coded queue and decrease the number of packets to send
Trigger "has source packets"
buffer of coded packets is empty
Code packets
There are coded packets in the coded queue
Send
Put the packet in the coded buffer and extract the oldest packet of the buffer

Figure 4.5: Process of what the recoder does when it receives a packet.

As a consequence of the issues the recoder was having because the coefficients created in every recoder were the same, the use of the updated library of kodo, *kodo-rlnc-c*, was taken into consideration. The implementation of the kodo-rlnc-c library in nc_kernel was asked to be done. Therefore, the testing of the recoder using the new version of kodo could then be done by incorporing the recoder to the new commands of kodo. Instead of using *kodoc_* the prefix used is *krlnc_*. By using this library, the recoders are able to set the seed to generate the coefficients. An important change to *kodo-c* is the disappearence of *on_the_fly* setting of coding, which is used when creating an encoder and recoder. *Kodo-rlnc-c* uses only three configurations of the coders, these are: *full_vector*, *seed* and *sparse_seed*. Online coding is done in all of the three configurations, the difference between them is the header the library uses in order to give the information of the coefficients used. The configuration the recoder will be using in the PACEMG protocol is *full_vector* encoding as the coefficients are sent as a vector, whereas in the other two, the seed used to generate the coefficients is sent instead. In order to assure the randomness between generations, whenever a new container is created, a new ranom seed is set to the coder of the container by using *krlnc_decoder_set_seed(coder, uint32_t seed)*. The flag of a systematic packet in *kodo-rknc-c* is not $0xFF$ anymore, the indicator that tells if it is a systematic packet or not, is the flag $0x02$ situated in the first byte of the kodo packet. In order to adapt the systematic recoder to the new *nc_kernel* library, the only change that had to be done was where to check if it was a systematic flag or not. Additional to this change, the recoders generate a random number every new generation and the seed used to produced the coefficient vectors is established, ensuring a randomness in all the generations and coders. All the rest of the recoder is the same as the one using *kodo-c*.

4.2 Simulation setup

With the purpose of testing the correct performance of the protocol, a simulation has been created. The Simulator is a c program that emulates a communication between nodes through a chain network. The intermediate nodes are recoders and the two external nodes are the encoder and decoder. Despite the fact that this protocol is intended to be used in mesh networks to be applied together with multipath communication, this simulation is done in a train-network because the purpose of the protocol is to be able to manage the arrival of unconsecutive packets of different generations and, the first step to test this, is to be able to manage this conduct before applying a multipath routing protocol. Therefore, this behaviour can also be emulated in a simpler network such as a type-train-nodes.
The collection of L links connecting $L + 1$ nodes in tandem is called *a line network of length* L i.e., every $l_i \in L$ is a direct link (n_i, n_{i+1}) between two nodes n_i and n_{i+1} , each two consecutive nodes are called the *sender or transmitting* and the *receiving* nodes respectively. Over this line network, a unicast problem (one source - one destination) is defined as follows: node n_1 is the source node (the encoder), that generates some data *p packets* and is responsible of encoding and sending the information to the next node. The rest of the intermediate nodes, also called *internal nodes*, are responsible of recoding and forwarding the message until the message reaches the sink node, the decoder, that will decode the information. The recoders also perform decoding to represent the behaviour of a sub-line-network smaller than the one emulated. For instance, when the recoder of n_3 decodes, it will approach to a 3 node train-network where n_3 is a decoder.

The inputs of the program simulator are in the Table 4.1. The parameter ”-j” is the jitter which is the value that makes the packets arrive at different times. A random value between 0 and this number is added to the latency every time a packet is sent with the intention of having different delay values for each packet. If the last packet from generation 1 takes 10 time slots (latency) + 10 time slots (jitter) to arrive to the next node, perhaps a packet from generation 2 will take 10 time slots (latency) + 0 time slots (jitter) to arrive at the next node, therefore, a packet from another generation will arrive before a packet from the previous generation. Note that the latency used in the sumulations is fixed (10 time slots).

The program writes in a file the number of packets decoded by every node and, in another separate file (*log file*), whenever a node decodes a packet, it is written the sequence number of the packet, the difference between the time the packet was created by the encoder and the decoding time, and these two values additionally. When losses are produced, at every lost packet, the link number and who was the packet addressed to it is also written.

Figure 4.6 represents the scenario of the simulator. The encoder creates and sends the packets filled with ranodm data every 1 time slot, when the recoders receive these packets they recode and send the new coded packets to the next node until the coded data reach the destination node (the decoder). The recoders not only recode the packets but also decode them to simulate an scenario where the destination node is each of the intermediate nodes.

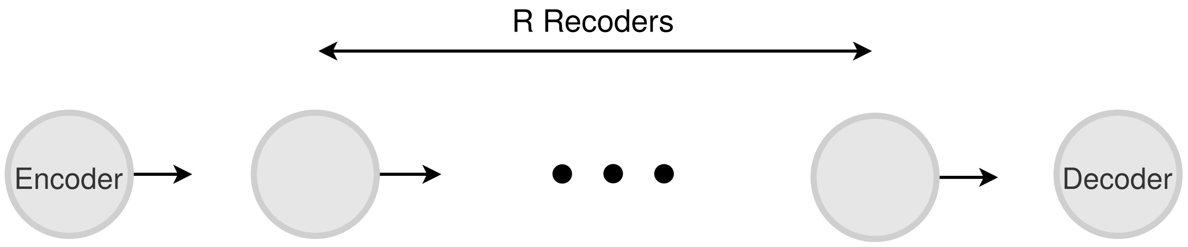


Figure 4.6: Simulation scenario

Variables	**Definition**
R	Number of recoders (nodes in the middle). If ”R = 3” there will be one encoder, three recoders and one decoder, hence 5 nodes.
-c	Coding Ratio of the coders (encoder and recoders)
-C	Number of maximum containers that are able to be managed simultaneously in the coders.
-s	Symbol Size (usually 1500 bytes)
-S	Symbols: number of source symbols in each generation.
-t	Tail Packets: number of coded packets put in the end of the generation to add redundancy.
-f	Feedback: if the feedback is on, every sent packet will have the feedback flag on (node will feedbak for every received packet).
-p	Number of packets to create and send
-j	Jitter: Number of time-slots that the packets are supposed to vary. If a packet is created every 1 time slot, and it takes a fixed value of 10 time slots to send a packet, we can add a ”-j” times lots. The bigger the jitter is, the more disordered the packets arrive at the next node. The amount of time slots to add to the latency is a random number between 0 and the ”-j” value in every transmission.
-e	Error rate: The maximum rate of losses in the link (%). With this probability a packet will be dropped in every link.
-l	Latency: fixed delay time from the sending of a packet until the arrival at the next node.
-T	Timeout Flush: timeout since the last received packet to flush the coder.
-F	File name to save the log data

Table 4.1: Input variables for the simulator

Chapter 5

Results

Some simulations were performed in order to test the behaviour of the protocol. In this section the results of the simulations are presented. As there are four versions of the recoder pacemg, the testing is done in the four of them. The performance is tested in a lossy network where the error rate changes (Section 5.1) and in a network where packets arrive disordered (Section 5.2).

5.1 In a lossy network: testing the feedback performance

PACEMG has been tested in a *simulation network* where the error rate goes from 0% to 50% with different coding ratios and various values of the maximum number of containers that the recoders could handle simultaneously. These simulations have been done with feedback and without. All the suimulations have been done in a network of length $L = 11$ which consists of one encoder, 10 recoders and one decoder. It takes 10 time slots for each packet to go from one node to the next one ($latency = 10timeslots$). Two hundred simmulations have been run for every changed input in order to have a consistent result. In the two sections below, some simmulation results when using feedback and without feedback will be shown. These are done in the systematic version of the recoder using kodo-rlnc-c.

5.1.1 Without feedback

The performance of the protocol when errors are present in the communications is tested. When feedbacks are not used, the coding ratio must be bigger to provide more redundant information in case several data is unable to reach a node.

In Figure 5.1, the recoders using kodo-c are implemented without feedbacks. This is a clear exaple of how this approach for the recoder does not work properly. When error rate is 0% and no losses and no jitter are present, the farer the nodes are from the encoder, the less infromation they can recover. This happens even though all sent packets are received. The reason for this is that PACEMG recoder $version_1$ using kodo-c faces an issue when the coding ratio is 100.

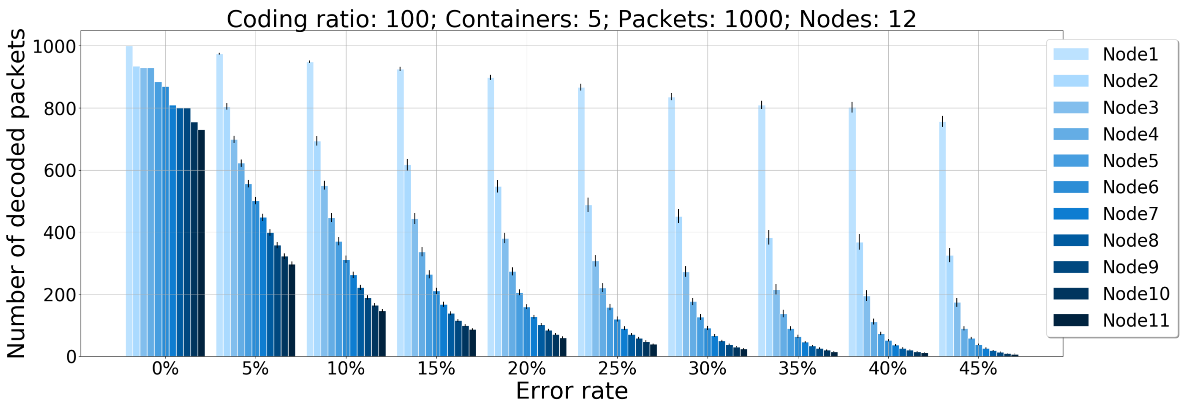
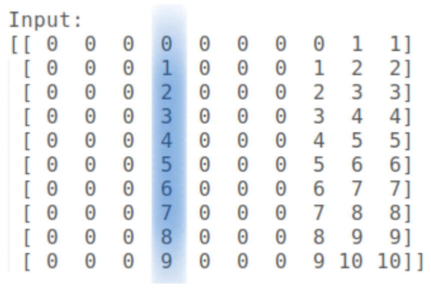


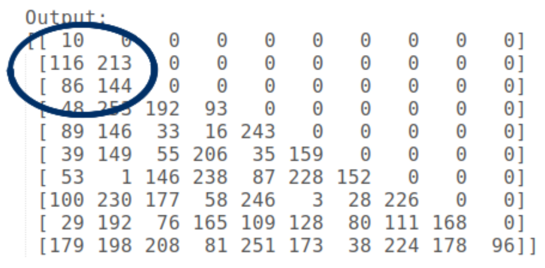
Figure 5.1: Example of the behaviour when using recoder *version 1* without feedbacks in a lossy network.

As said before, the coefficients are always the same and, as it can be seen in Figure 5.2, the 3rd packet taken out from the Kodo coder, has two coefficients instead of three (the 3rd line of the matrix) which means that only the two first source symbols are combined to form a coded packet (and not the third one). Therefore, when the receiver receives the first two packets, it updates the rank of the generation to 2, however, when the 3rd coded packet arrives, as it is the combination of $sourcesymbol_1$ and $sourcesymbol_2$, and not $sourcesymbol_3$, the rank is still 2. Moreover, the sequence of numbers of constants used to code in each recoder are the same as the seed employed to do so is also the same. As long as the containers are treated independently, which they are, the generated packets are independent within its generation. The problem lies here: as the recoders use the same seed and, whenever a recoder receives, for instance, a packet that was the second packet sent, which corresponds to the fourth packet next node receives, it will generate another coefficient which, thereafter, when next node receives it, and it is the second packet received, the node will generate the coefficients from the second posititon (the same as the first node). Dependency is caused when this happens to more than one packet and they mix coefficients by mixing arrival positions.



```
Input:
[[ 0  0  0  0  0  0  0  0  1  1]
 [ 0  0  0  1  0  0  0  1  2  2]
 [ 0  0  0  2  0  0  0  2  3  3]
 [ 0  0  0  3  0  0  0  3  4  4]
 [ 0  0  0  4  0  0  0  4  5  5]
 [ 0  0  0  5  0  0  0  5  6  6]
 [ 0  0  0  6  0  0  0  6  7  7]
 [ 0  0  0  7  0  0  0  7  8  8]
 [ 0  0  0  8  0  0  0  8  9  9]
 [ 0  0  0  9  0  0  0  9 10 10]]
```

The input come from the encoder and the packets are systematic. The colored numbers are the sequence number of the generation. The three numbers on the right are the payload of the packet. As it is a systematic packet, the header does not contain the coefficients, hence, it is smaller than a coded packet.



```
Output:
[[ 10   0   0   0   0   0   0   0   0   0]
 [116 213   0   0   0   0   0   0   0   0]
 [ 86 144   0   0   0   0   0   0   0   0]
 [ 48 255 192  93   0   0   0   0   0   0]
 [ 89 146  33  16 243   0   0   0   0   0]
 [ 39 149  55 206  35 159   0   0   0   0]
 [ 53   1 146 238  87 228 152   0   0   0]
 [100 230 177  58 246   3  28 226   0   0]
 [ 29 192  76 165 109 128  80 111 168   0]
 [179 198 208  81 251 173  38 224 178  96]]
```

The sent packets created from the inputted packets include the coefficients seen in the image. The first two rows correspond to (1) the first symbol and (2) the combination of the first and second symbol of the generation. By the time the third symbol is available in the coder, the coefficient vector only has two coefficients which combine again the two frist symbols, but not the third one. As it has coding ratio 100, next node will not have enough information to recover the original data.

Figure 5.2: Recoder version 1 using kodo-c. Coefficients received and generated by $node_1$ (first recoder) of generation 1 with coding ratio of 100.

5.1.2 With Feedback

The graphic below (Figure 5.3) shows the case when the feedback implementation in the recoders is the same as the one in the encoders. The amount of data recovered in the decoder (node 11) is higher than the decoded packets in the recoders. This is due to the fact that when an aknowledgement is received, the recoder, in this configuration, deletes all the information that de next node says it already has. Therefore, when a generation is not decodable, and a feedback packet arrives, that generation and some newer ones are deleted, making impossible in the recoder recover the information from the other deleted generations.

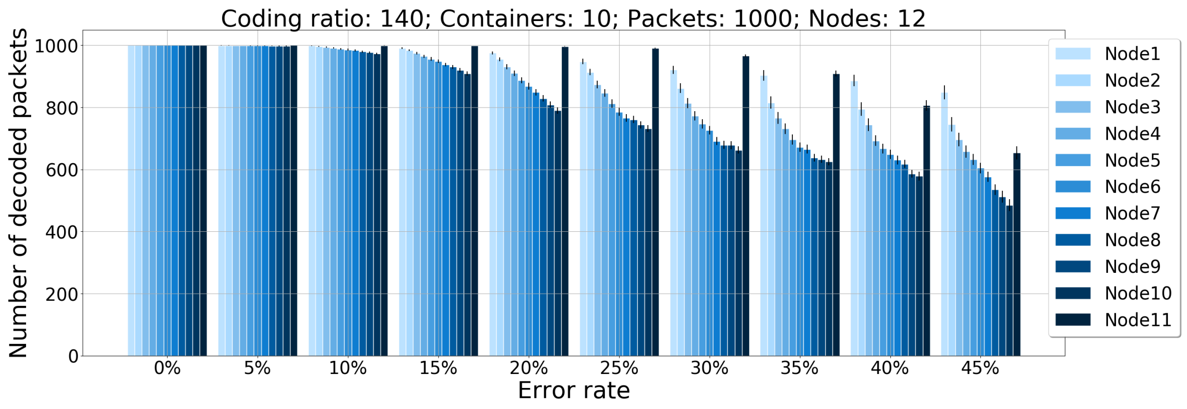


Figure 5.3: Example of the performance fo the network when the recoders manage feedbacks the same way as the encoders.

As NC is used with more than 100% of coding ratio in order to generate redundant information, when the coding ratio is increased, so is the received data in each of the nodes. Furthermore, when feedback is applied, the performance increases considerbaly. Clarify that, in order to be able to use ARQ, more than one container must be used since packet from other generations may arrive before the re-sent packets reach the node.

The next two graphics (Figure 5.4 and Figure 5.5) show the systematic recoder (*version 2*) using kodo-rlnc-c when feedback is used. In this set of simulations represented in the graphics, the feedback mechanism is the one that enables the recoder decode as the decoder would do. It is doubtless that more containers means more robustness and the protocol performs better when the number of containers is greater than when the number of redundant packets is larger. It must be pointed out that no jitter is added, therefore, when the coders have a wider range of generations to manage, the previous node can also re-send information of all those generations, instead of only a few (as in Figure 5.5).

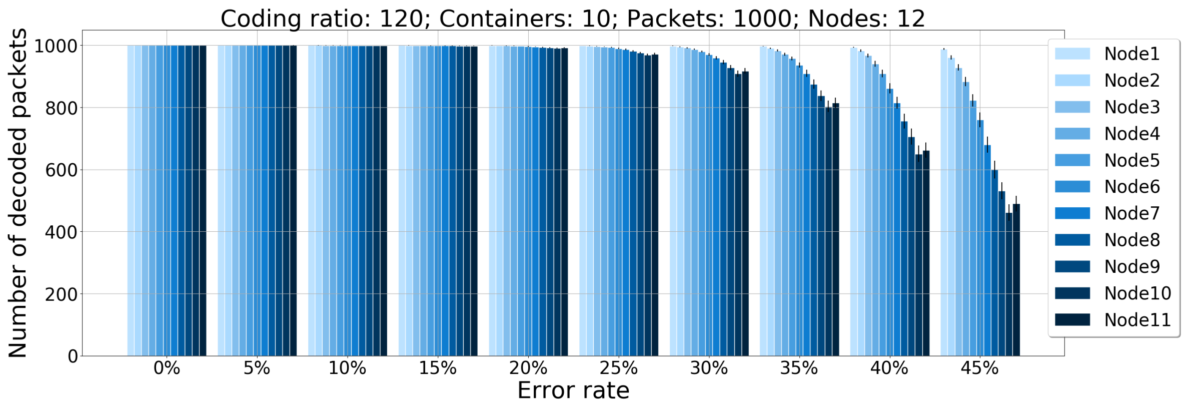


Figure 5.4: Graph of a set of simulations using systematic recoders with kodo-rlnc-c library (coding ratio 120% and 10 maximum containers).

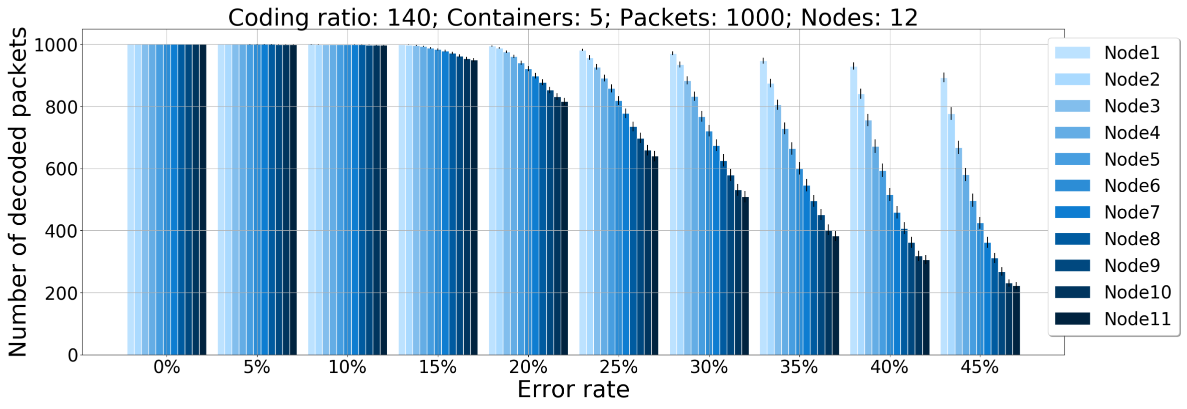


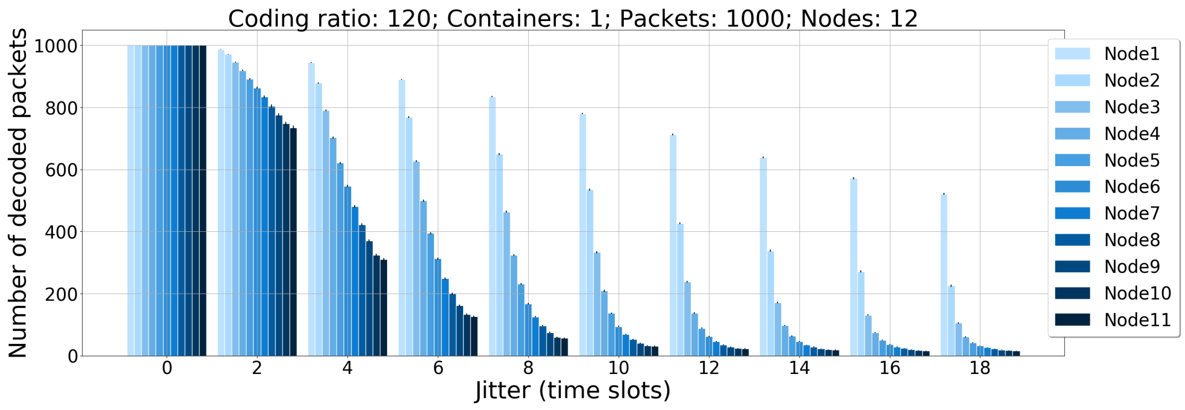
Figure 5.5: Graph of a set of simulations using systematic recoders with kodo-rlnc-c library (coding ratio 140% and 5 maximum containers).

5.2 Changing Jitter without feedback

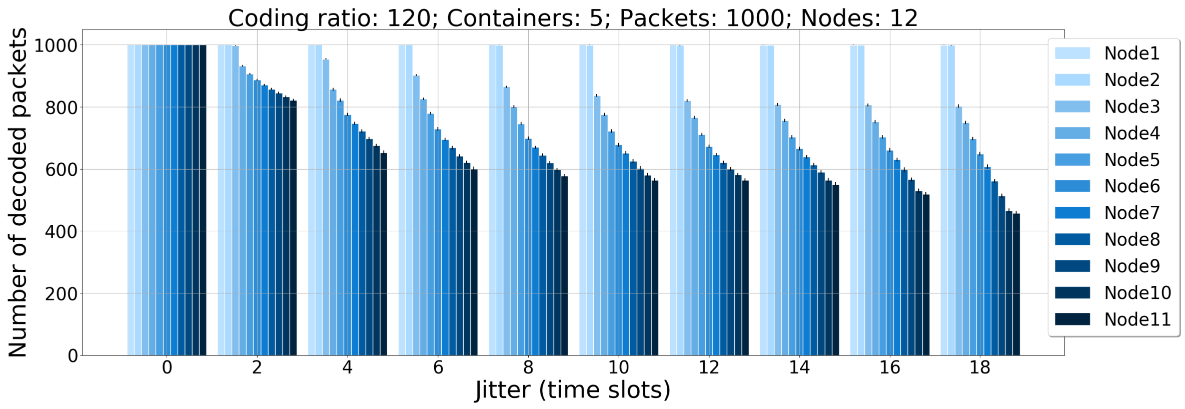
In multipah communications, packets travel through different paths and arrive disorderly. In this section this behaviour is shown in the four recoders.

5.2.1 Conventional recoder using kodo-c

The two figures below (Figure 5.6) show the improvement when nodes can handle more than one container. When the jitter gets close to 20 time slots, the decoded packets is minor as when the jitter is 10 since having 20 time slots of jitter means that a packet can reach the next node after 10 + 20 time slots from the sending time, but another one sent at the same time (from another generation currently available) can be received after 10 time slots inducing this highly unorganized arrival of packets. The achivement with one container is very poor compared to when more containers can be handled.



By using only one container, as new information from different containers reach the nodes, older generations may not continue receiving packets and must be delted.

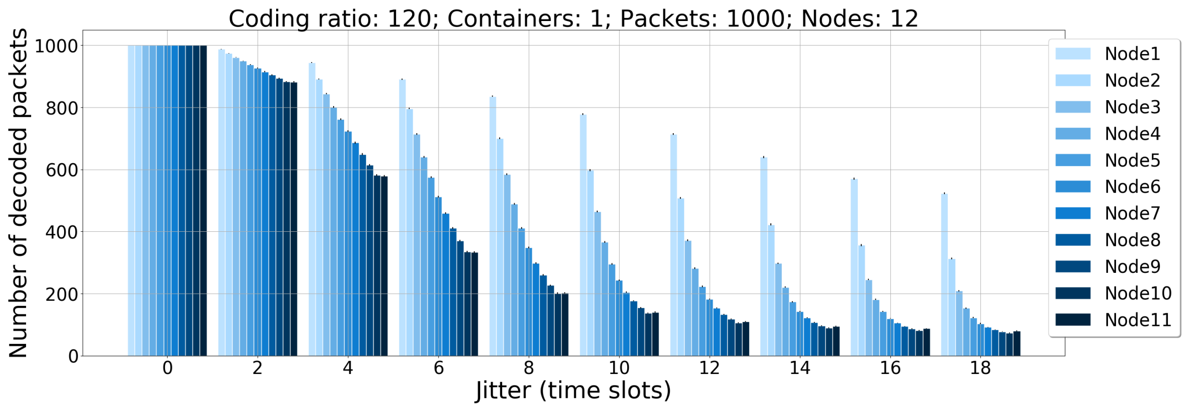


Five containers can be handled by the coders and the number of decoded packets increase significally when jitter is high in contrast to the previous Chart.

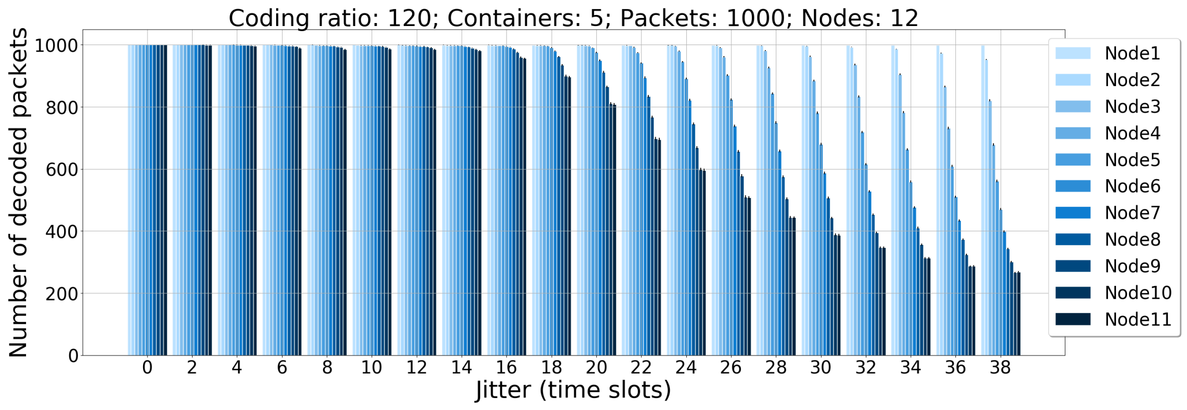
Figure 5.6: Decoded packets in every node using recoders *version 1* with kodo-c library.

5.2.2 Systematic recoder using kodo-c

The implementation of systematic recoding enhance the fucntioning of the protocol, as it is more probable to receive *innovative* packets. The problem regarding the dependency between packets is almost solved as the protocol tries to avoid generating coded packets. As long as the received packets are sent as they are, the coefficients taken from the kodo are not harmful to the decoding process.



Nodes can operate with a maximum of one container. The higher the jitter is, the less information the node can recover.



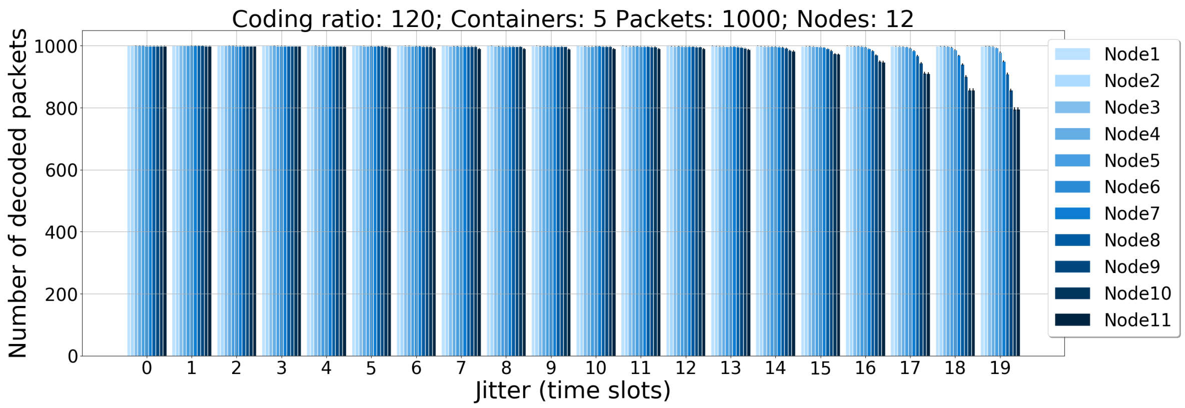
A significant increase of original symbols has been achieved by configuring nodes to manage 5 containers. Almost all the data is recovered in all the nodes when jitter is high.

Figure 5.7: Decoded packets in every node using systematic recoding (recoders *version 2*) with kodo-c library

These two results seen in Figure 5.7 also show the improvement of the implementation of multi-generation PACE protocol. By only adding until 5 simultaneous containers, the improvement is meaningful.

5.2.3 Recoder *version 1* using kodo-rlnc-c

The modification of a library that provides a method to set the seed values for the coders yields the same result as the previous systematic apporoach (systematic recoder *version 2* using kodo-c).

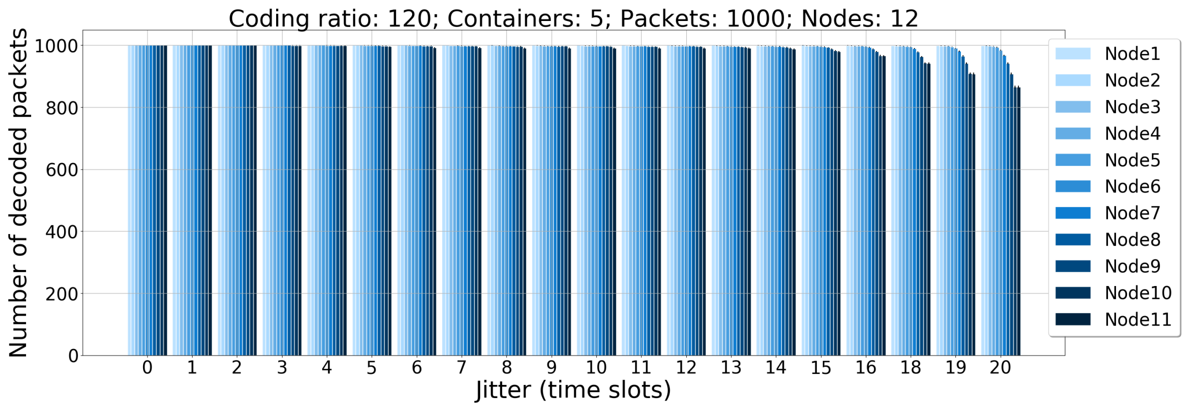


The performance when coders can have 5 containers functioning at the same time is satisfactory. Nodes can decode a large number of packets even when jitter is high.

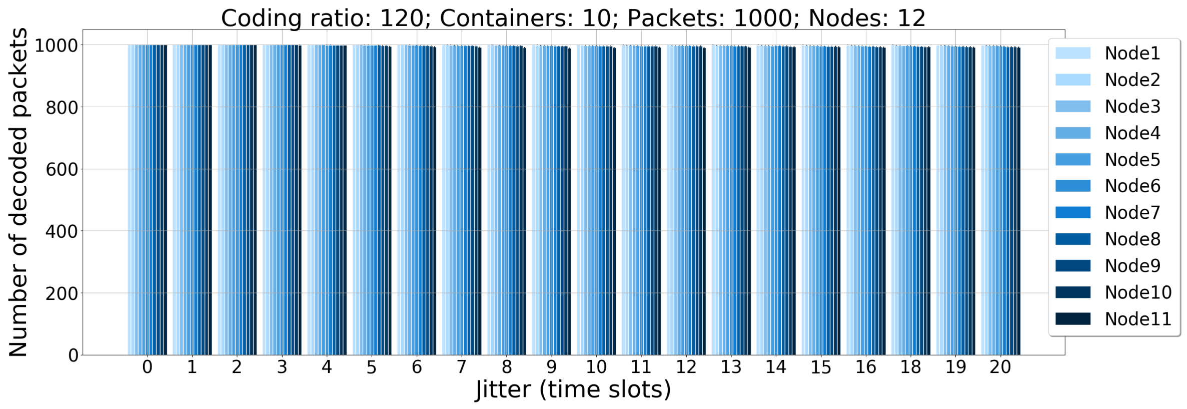
Figure 5.8: Decoded packets in every node using recoders *version 1* with kodo-rlnc-c library

5.2.4 Systematic recoder using kodo-rlnc-c

The testing of the last version of the recoder with non-sequential arrival of packets is below (Figrue 5.9). It can be stated that this version of the recoder of PACEMG has a better achievement compared to the other three versions of the recoder. When the maximum number of contaienrs reach 10, the nodes decode almost all the information.



Although the packets arrive wiht highly messiness (non-sequentially), recoders can still decode a substantial amount of data.



If nodes can manage until 10 containers, the original information is recovered in each of the nodes

Figure 5.9: Decoded packets in every node using systematic recoding (recoders *version 2*) with kodo-rlnc-c library

Chapter 6

Conclusions and future work

6.1 Conclusions

In this thesis a protocol called PACEMG that supports multiple generations at the same time was improved. A recoder was developed and tested in different settings. This recoder module, that can relay messages and obtain coded information, is essential for implementing a RLNC protocol in multi-hop communications. It can act as both, an encoder and a decoder. In the process of the creation of the recoder, some challenges have presented themselves

The first and most important one is the dependency caused by using the same coefficients in different coders. Using the same seed in every coder for the same generations may cause dependencies between coded packets. Suppose the packet that is generated from $node_1$ is the 3rd packet to be generated and node 2 receives this packet once it has 4 linearly independent packets in the coder. The coefficient vector used to generate the 4th packet in Node 2 is generating the same coefficients as the 4th packet generated in $node_1$. If it is assumed that the outputted packet from Node 2 is the second packet received in Node 3, Node 3 will be generating the same coefficients as $node_1$ had. If that happens to more packets of the same generation, dependencies between packets from that generation can be caused, provoking the decoder not to be able to decode. Consecutive coders should not use the same coefficients for the same generation when packets arrive disordered.

This challenge has been solved by not allowing the coder to send all the information as coded packets but a combination of systematic and coded packets. The dependency between symbols can be avoided if all the systematic packets that arrive at a node are sent without coding, causing that the number of independent sent packets is larger. However, as a consequence of the increase of the number of nodes and presence of losses, systematic packets can get lost provoking the necessity of generating and sending more coded packets. If that happens, the recoders, taking into account that they still create the same coefficients for the same generations, would be facing the same issues as the recoder *version 1*.

By changing a library to enable the adjustment the seed, this problem is overcome. However, the use of a systematic recoder is still necessary as the purpose of this protocol is a short-delay multipath RLNC protocol. If systematic re-coding is not performed, the enocder, which performs PACE encoding (locate the redundant packets dispersed within the generation in order to reduce the per-packet delay) is unnecessary as the decoder can not perform instant decoding because the recoders have only sent coded information. It has been seen that, the best performance of the recoder is done by the systematic recoder using a kodo-rlnc-c library. It has been shown experimentally that this enhanced protocol significantly improves the throughput over the PACE protocol (PACEMG one-generation) under the influence of errors and jitters. Moreover, when feedback messages are used, the reliability of the tranmission is higher when the coders can handle more containers.

6.2 Future Work

In the following points research directions are proposed that could be investigated as an extension to the research presented in this thesis.

(a) Error and Jitter testing. More simulations could be done mixing the loss probability with the jitter as in this thesis, the jitter and errors have been tested independently.

(b) PACEMG together with OpR in a mesh network: In this thesis the functioning of PACEMG protocol has been tested when packets get lost and when the receiving of the information is such disorganized that it needs to have more than one generation to be able to work propoerly. The next approach could be to test it in a mesh network and, to do so, OpR is needed.

(c) Coding Ratio adaptable: Currently, the coding ratio of a coder is set when it is created. However, the errors in the network may vary and the coding ratio should too in order to ensure, in case there are multiple lost packets, the successfull receiving (decoding) of the information. To achieve this, the part of the recoder that saves the positions of the coded packets that should be sent after some systematic ones, which, at the moment, is located in the coder and is shared by all the containers, should be located in each of the containers. Because, if a container had coding ratio 120% and the next one changed to 160% the order of the coded symbols sending should change.

Bibliography

[1] S. Pandi, F. Gabriel, J. A. Cabrera, S. Wunderlich, M. Reisslein, and F. H. P. Fitzek. Pace: Redundancy engineering in rlnc for low-latency communication. *IEEE Access*, 5:20477–20493, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2736879.

[2] R. Ahlswede, Ning Cai, S. Y.R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inf. Theor.*, 46(4):1204–1216, September 2006. ISSN 0018-9448. doi: 10.1109/18.850663. URL `https://doi.org/10.1109/18.850663`.

[3] Cisco White Paper. Cisco visual networking index: Global mobile data traffic forecast update, 2010-2015. February 2011. URL `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf`.

[4] M. Ghaderi, D. Towsley, and J. Kurose. Reliability gain of network coding in lossy wireless networks. In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pages 2171–2179, April 2008. doi: 10.1109/INFOCOM.2008.284.

[5] E. Tsimbalo, A. Tassi, and R. J. Piechocki. Reliability of multicast under random linear network coding. *IEEE Transactions on Communications*, 66(6):2547–2559, June 2018. ISSN 0090-6778. doi: 10.1109/TCOMM.2018.2801791.

[6] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. Xors in the

air: Practical wireless network coding. *IEEE/ACM Transactions on Networking*, 16(3):497–510, June 2008. ISSN 1063-6692. doi: 10.1109/TNET.2008.923722.

[7] D. Lun, M. Médard, R. Koetter, and M. Effros. On coding for reliable communication over packet networks. *Physical Communication*, 1:3–20, 09 2005. doi: 10.1016/j.phycom.2008.01.006.

[8] A. Eryilmaz, A. Ozdaglar, and M. Medard. On delay performance gains from network coding. In *2006 40th Annual Conference on Information Sciences and Systems*, pages 864–870, March 2006. doi: 10.1109/CISS.2006.286588.

[9] L. Lima, J. Vilela, J. Barros, and M. Medard. An information-theoretic cryptanalysis of network coding - is protecting the code enough? pages 1 – 6, 01 2009. doi: 10.1109/ISITA.2008.4895420.

[10] L. Lima, M. Medard, and J. Barros. Random linear network coding: A free cipher? In *2007 IEEE International Symposium on Information Theory*, pages 546–550, June 2007. doi: 10.1109/ISIT.2007.4557282.

[11] S. Feizi, D. E. Lucani, C. W. Sørensen, A. Makhdoumi, and M. Médard. Tunable sparse network coding for multicast networks. In *2014 International Symposium on Network Coding (NetCod)*, pages 1–6, June 2014. doi: 10.1109/NETCOD.2014.6892129.

[12] B. Shrader and N. M. Jones. Systematic wireless network coding. In *MILCOM 2009 - 2009 IEEE Military Communications Conference*, pages 1–7, Oct 2009. doi: 10.1109/MILCOM.2009.5380081.

[13] I. Chatzigeorgiou and C. Price. Random linear network coding for satellite-aided flight data streaming. 07 2018. doi: 10.1109/PIMRC.2018.8580972.

[14] N. Papanikos and E. Papapetrou. Deterministic broadcasting and random linear network coding in mobile ad hoc networks. *IEEE/ACM Transactions on*

Networking, 25(3):1540–1554, June 2017. ISSN 1063-6692. doi: 10.1109/TNET.2016.2641680.

[15] F. H. P. Fitzek and M.D. Katz. *Mobile Clouds: Exploiting Distributed Resources in Wireless, Mobile and Social Networks.* Wiley, 2013. ISBN 9781118801444. URL `https://books.google.es/books?id=s2lXAgAAQBAJ`.

[16] M. Gumel and N. Faruk. Wireless mesh networks throughput capacity analysis. *Nigeria Journal of Technological Development, Faculty of Engineering and technology, University of Ilorin, Ilorin, Nigeria*, 9:37–46, 12 2012.

[17] P. Jadhav and R. Satao. A survey on opportunistic routing protocols for wireless sensor networks. *Procedia Computer Science*, 79:603–609, 12 2016. doi: 10.1016/j.procs.2016.03.076.

[18] E. Rozner, J. Seshadri, Y. Mehta, and L. Qiu. Soar: Simple opportunistic adaptive routing protocol for wireless mesh networks. *IEEE Transactions on Mobile Computing*, 8(12):1622–1635, Dec 2009. ISSN 1536-1233. doi: 10.1109/TMC.2009.82.

[19] E. Rozner, J. Seshadri, Y. Mebta, and L. Qiu. Simple opportunistic routing protocol for wireless mesh networks. In *2006 2nd IEEE Workshop on Wireless Mesh Networks*, pages 48–54, Sep. 2006. doi: 10.1109/WIMESH.2006.288602.

[20] D. Gómez, P. Garrido, E. Rodríguez, R. Agüero, and L. Muñoz. Enhanced opportunistic random linear source/network coding with cross-layer techniques over wireless mesh networks. In *2014 IFIP Wireless Days (WD)*, pages 1–4, Nov 2014. doi: 10.1109/WD.2014.7020842.

[21] B. Ni, N. Santhapuri, Z. Zhong, and S. Nelakuduti. Routing with opportunistically coded exchanges in wireless mesh networks. In *2006 2nd IEEE Workshop on Wireless Mesh Networks*, pages 157–159, Sep. 2006. doi: 10.1109/WIMESH.2006.288636.

[22] P. Pahlevani, D. E. Lucani, M. V. Pedersen, and F. H. P. Fitzek. Playncool: Opportunistic network coding for local optimization of routing in wireless mesh networks. In *2013 IEEE Globecom Workshops (GC Wkshps)*, pages 812–817, Dec 2013. doi: 10.1109/GLOCOMW.2013.6825089.

[23] D. Koutsonikolas, C. Wang, and Y. C. Hu. Ccack: Efficient network coding based opportunistic routing through cumulative coded acknowledgments. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, March 2010. doi: 10.1109/INFCOM.2010.5462125.

[24] Q. Hu and J. Zheng. Coaor: An efficient network coding aware opportunistic routing mechanism for wireless mesh networks. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 4578–4583, Dec 2013. doi: 10.1109/GLOCOMW.2013.6855673.

[25] S. Biswas and R. Morris. Exor: Opportunistic multi-hop routing for wireless networks. volume 35, pages 133–144, 10 2005. doi: 10.1145/1080091.1080108.

[26] J. Widmer and J-Y. Le Boudec. Network coding for efficient communication in extreme networks. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking*, WDTN '05, pages 284–291, New York, NY, USA, 2005. ACM. ISBN 1-59593-026-4. doi: 10.1145/1080139.1080147. URL `http://doi.acm.org/10.1145/1080139.1080147`.

[27] X. Zhang, G. Neglia, J. Kurose, and D. Towsley. On the benefits of random linear coding for unicast applications in disruption tolerant networks. In *2006 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 1–7, Feb 2006. doi: 10.1109/WIOPT.2006.1666505.

[28] Srihari Nelakuditi and Zhi-Li Zhang. On selection of paths for multipath routing. In *IWQoS*, 2001.

[29] J. W. Tsai and T. Moors. A review of multipath routing protocols : From wireless ad hoc to mesh networks. 2006.

[30] A. Ghaffari and S. Babazadeh. Multi-path routing based on network coding in wireless sensor networks. *World Applied Sciences Journal*, 21:1657–1663, 01 2013. doi: 10.5829/idosi.wasj.2013.21.11.1826.

[31] A. Eryilmaz, A. Ozdaglar, M. Medard, and E. Ahmed. On the delay and throughput gains of coding in unreliable networks. *IEEE Transactions on Information Theory*, 54(12):5511–5524, Dec 2008. ISSN 0018-9448. doi: 10.1109/TIT.2008.2006454.

[32] X. Li, C. Wang, and X. Lin. Throughput and delay analysis on uncoded and coded wireless broadcast with hard deadline constraints. In *2010 Proceedings IEEE INFOCOM*, pages 1–5, March 2010. doi: 10.1109/INFCOM.2010.5462258.

[33] P. Parag and J. Chamberland. Queueing analysis of a butterfly network for comparing network coding to classical routing. *IEEE Transactions on Information Theory*, 56(4):1890–1908, April 2010. ISSN 0018-9448. doi: 10.1109/TIT.2010.2040862.

[34] L. Nielsen, R. Rydhof Hansen, and D. E. Lucani. Latency performance of encoding with random linear network coding. In *European Wireless 2018; 24th European Wireless Conference*, pages 1–5, May 2018.

[35] E. Drinea, C. Fragouli, and L. Keller. Real-time delay with network coding and feedback.

[36] V. Roca, B. Teibi, C. Burdinat, T. Tran-Thai, and C. Thieno. Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications. working paper or preprint, February 2017. URL `https://hal.inria.fr/hal-01395937`.

[37] J. Qureshi, C. Foh, and J. Cai. Online xor packet coding: Efficient single-hop wireless multicasting with low decoding delay. *Computer Communications*, 39, 01 2013. doi: 10.1016/j.comcom.2013.09.006.

[38] S. Wunderlich, F. Gabriel, S. Pandi, and F. H. P. Fitzek. We don't need no generation - a practical approach to sliding window rlnc. In *2017 Wireless Days*, pages 218–223, March 2017. doi: 10.1109/WD.2017.7918148.

[39] A. Garcia-Saavedra, M. Karzand, and D. J. Leith. Low delay random linear coding and scheduling over multiple interfaces. *IEEE Transactions on Mobile Computing*, 16(11):3100–3114, Nov 2017. ISSN 1536-1233. doi: 10.1109/TMC.2017.2686379.

[40] J. Heide, M. Videbæk Pedersen, and F. H. P. Fitzek. Decoding algorithms for random linear network codes. pages 129–136, 05 2011. doi: 10.1007/978-3-642-23041-7_13.

[41] C. Gkantsidis and P. R. Rodriguez. Network coding for large scale content distribution. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 4, pages 2235–2245 vol. 4, March 2005. doi: 10.1109/INFCOM.2005.1498511.

[42] K. Jain, L. Lovász, and P. Chou. Building scalable and robust -to- overlay networks for broadcasting using network coding. *Distributed Computing*, 19:301–311, 03 2007. doi: 10.1007/s00446-006-0014-9.

[43] C. Wu and B. Li. Echelon: -to- network diagnosis with network coding. In *200614th IEEE International Workshop on Quality of Service*, pages 20–29, June 2006. doi: 10.1109/IWQOS.2006.250447.

[44] B. Li and D. Niu. Random network coding in -to- networks: From theory to practice. *Proceedings of the IEEE*, 99(3):513–523, March 2011. ISSN 0018-9219. doi: 10.1109/JPROC.2010.2091930.

[45] N. Aboutorab, P. Sadeghi, and S. Sorour. Enabling a tradeoff between completion time and decoding delay in instantly decodable network coded systems. *IEEE Transactions on Communications*, 62(4):1296–1309, April 2014. ISSN 0090-6778. doi: 10.1109/TCOMM.2014.021614.130172.

[46] A. Douik, M. S. Karim, P. Sadeghi, and S. Sorour. Delivery time reduction for order-constrained applications using binary network codes. In *2016 IEEE Wireless Communications and Networking Conference*, pages 1–6, April 2016. doi: 10.1109/WCNC.2016.7565067.

[47] A. Douik, S. Sorour, T. Y. Al-Naffouri, and M. Alouini. Instantly decodable network coding: From centralized to device-to-device communications. *IEEE Communications Surveys Tutorials*, 19(2):1201–1224, Secondquarter 2017. ISSN 1553-877X. doi: 10.1109/COMST.2017.2665587.

[48] S. Sorour and S. Valaee. Completion delay minimization for instantly decodable network codes. *CoRR*, abs/1201.4768, 2012. URL `http://arxiv.org/abs/1201.4768`.

[49] M. Yu, N. Aboutorab, and P. Sadeghi. From instantly decodable to random linear network coded broadcast. *IEEE Transactions on Communications*, 62(11): 3943–3955, Nov 2014. ISSN 0090-6778. doi: 10.1109/TCOMM.2014.2364198.

[50] X. Li, C. Wang, and X. Lin. Optimal immediately-decodable inter-session network coding (idnc) schemes for two unicast sessions with hard deadline constraints. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 784–791, Sep. 2011. doi: 10.1109/Allerton.2011.6120247.

[51] A. Douik, S. Sorour, M. Alouini, and T. Y. Al-Naffouri. Completion time reduction in instantly decodable network coding through decoding delay control. In *2014 IEEE Global Communications Conference*, pages 5008–5013, Dec 2014. doi: 10.1109/GLOCOM.2014.7037599.

[52] J. K. Sundararajan, D. Shah, and M. Medard. Online network coding for optimal throughput and delay - the three-receiver case. In *2008 International Symposium on Information Theory and Its Applications*, pages 1–6, Dec 2008. doi: 10.1109/ISITA.2008.4895447.

[53] J. K. Sundararajan, P. Sadeghi, and M. Medard. A feedback-based adaptive broadcast coding scheme for reducing in-order delivery delay. In *2009 Workshop on Network Coding, Theory, and Applications*, pages 1–6, June 2009. doi: 10.1109/NETCOD.2009.5437470.

[54] Y. Lin, B. Liang, and B. Li. Slideor: Online opportunistic network coding in wireless mesh networks. In *2010 Proceedings IEEE INFOCOM*, pages 1–5, March 2010. doi: 10.1109/INFCOM.2010.5462249.

[55] Y. Gao, N. Zhang, and G. Kang. A novel online network coding scheme for broadcast channels with imperfect feedback and decoding delay constraints. In *2018 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 448–453, Aug 2018. doi: 10.1109/ICCChina.2018.8641246.

[56] D. E. Lucani, M. Medard, and M. Stojanovic. Online network coding for time-division duplexing. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6, Dec 2010. doi: 10.1109/GLOCOM.2010.5683892.

[57] I. Talzi and C. Tschudin. Online code compression in wireless sensor networks. In *2011 International Conference on Collaboration Technologies and Systems (CTS)*, pages 301–310, May 2011. doi: 10.1109/CTS.2011.5928702.

[58] S. Yang, X. Wang, and B. Li. Haste: Practical online network coding in a multicast switch. In *2010 Proceedings IEEE INFOCOM*, pages 1–5, March 2010. doi: 10.1109/INFCOM.2010.5462194.

[59] P. A. Chou and Y. Wu. Practical network coding. 10 2003.

[60] P. A. Chou and Y. Wu. Network coding for the internet and wireless networks. *IEEE Signal Processing Magazine*, 24(5):77–85, Sep. 2007. ISSN 1053-5888. doi: 10.1109/MSP.2007.904818.

[61] I. Chatzigeorgiou and A. Tassi. Decoding delay performance of random linear network coding for broadcast. *IEEE Transactions on Vehicular Technology*, 66 (8):7050–7060, Aug 2017. ISSN 0018-9545. doi: 10.1109/TVT.2017.2670178.

[62] J. Claridge and I. Chatzigeorgiou. Probability of partially decoding network-coded messages. *IEEE Communications Letters*, 21(9):1945–1948, Sep. 2017. ISSN 1089-7798. doi: 10.1109/LCOMM.2017.2704110.

[63] G. Cocco, T. de Cola, and M. Berioli. Performance analysis of queueing systems with systematic packet-level coding. In *2015 IEEE International Conference on Communications (ICC)*, pages 4524–4529, June 2015. doi: 10.1109/ICC.2015.7249035.

[64] M. Kwon and H. Park. Analysis on decoding error rate of systematic network coding. In *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pages 258–259, Jan 2017. doi: 10.1109/ICCE.2017.7889308.

[65] B. Shrader and A. Ephremides. Queueing delay analysis for multicast with random linear coding. *IEEE Transactions on Information Theory*, 58(1):421–429, Jan 2012. ISSN 0018-9448. doi: 10.1109/TIT.2011.2171516.

[66] Y. Li, E. Soljanin, and P. Spasojevic. Effects of the generation size and overlap on throughput and complexity in randomized linear network coding. *IEEE Transactions on Information Theory*, 57(2):1111–1123, Feb 2011. ISSN 0018-9448. doi: 10.1109/TIT.2010.2095111.

[67] M. Nistor, D. E. Lucani, T. T. V. Vinhoza, R. A. Costa, and J. Barros. On the delay distribution of random linear network coding. *IEEE Journal on Selected Areas in Communications*, 29(5):1084–1093, May 2011. ISSN 0733-8716. doi: 10.1109/JSAC.2011.110518.

[68] A. Alamdar Yazdi, S. Sorour, S. Valaee, and R. Y. Kim. Optimum network coding for delay sensitive applications in wimax unicast. In *IEEE INFOCOM 2009*, pages 2576–2580, April 2009. doi: 10.1109/INFCOM.2009.5062190.

[69] W. Zeng, C. T. K. Ng, and M. Médard. Joint coding and scheduling optimization in wireless systems with varying delay sensitivities. *CoRR*, abs/1202.0784, 2012. URL `http://arxiv.org/abs/1202.0784`.

[70] M. Yu and P. Sadeghi. Approximating throughput and packet decoding delay in linear network coded wireless broadcast. *CoRR*, abs/1701.04551, 2017. URL `http://arxiv.org/abs/1701.04551`.

[71] A. Douik, S. Sorour, T. Y. Al-Naffouri, H. Yang, and M. Alouini. Delay reduction in multi-hop device-to-device communication using network coding. In *2015 International Symposium on Network Coding (NetCod)*, pages 6–10, June 2015. doi: 10.1109/NETCOD.2015.7176779.

[72] K. Prasad and B. S. Rajan. On network coding for acyclic networks with delays. In *2011 IEEE Information Theory Workshop*, pages 523–527, Oct 2011. doi: 10.1109/ITW.2011.6089517.

[73] K. Prasad and B. S. Rajan. Single-generation network coding for networks with delay. In *2010 IEEE International Conference on Communications*, pages 1–6, May 2010. doi: 10.1109/ICC.2010.5502118.

[74] R. Torre, S. Pandi, G. T. Nguyen, and F. H. P. Fitzek. Optimization of a random linear network coding system with newton method for wireless systems. In *2019 IEEE International Conference on Communications (ICC): Communication QoS, Reliability and Modeling Symposium*, Shanghai, China, 2019.

[75] J. Rischke, F. Gabriel, S. Pandi, G. T. Nguyen, H. Salah, and F. H. P. Fitzek. Improving communication reliability efficiently: Adaptive redundancy for rlnc in sdn. In *2019 IEEE Conference on Network Softwarization (NetSoft) (NetSoft 2019)*, Paris, France, 2019.

[76] S. Wunderlich, F. Gabriel, S. Pandi, F. H. P. Fitzek, and M. Reisslein. Caterpillar rlnc (crlnc): A practical finite sliding window rlnc approach. *IEEE Access*, 5: 20183–20197, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2757241.

[77] F. Gabriel, A. K. Chorppath, I. Tsokalo, and F. H. P. Fitzek. Multipath communication with finite sliding window network coding for ultra-reliability and low

latency. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, May 2018. doi: 10.1109/ICCW.2018.8403489.

[78] S. Wunderlich, F. Gabriel, S. Pandi, F. H. P. Fitzek, and M. Reisslein. Caterpillar rlnc (crlnc): A practical finite sliding window rlnc approach. *IEEE Access*, 5: 20183–20197, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2757241.

[79] F. Gabriel, S. Wunderlich, S. Pandi, F. H. P. Fitzek, and M. Reisslein. Caterpillar rlnc with feedback (crlnc-fb): Reducing delay in selective repeat arq through coding. *IEEE Access*, 6:44787–44802, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2865137.

[80] N. Cai and R.W. Yeung. Network error correction, ii: Lower bounds. *Communications in Information and Systems*, 6, 01 2006. doi: 10.4310/CIS.2006.v6.n1.a3.

[81] N. Cai and R.W. Yeung. Network coding and error correction. pages 119 – 122, 11 2002. ISBN 0-7803-7629-3. doi: 10.1109/ITW.2002.1115432.

[82] S. Y.R. Li, R. W. Yeung, and Ning Cai. Linear network coding. *IEEE Trans. Inf. Theor.*, 49(2):371–381, February 2003. ISSN 0018-9448. doi: 10.1109/TIT.2002.807285. URL `https://doi.org/10.1109/TIT.2002.807285`.

[83] R. Koetter and M. Medard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11(5):782–795, Oct 2003. ISSN 1063-6692. doi: 10.1109/TNET.2003.818197.

[84] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *IEEE International Symposium on Information Theory, 2003. Proceedings.*, pages 442–, June 2003. doi: 10.1109/ISIT.2003.1228459.

[85] T. Ho, M. Medard, R. Koetter, D.R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52:4413–4430, 2006.

[86] M. Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 271–280, Nov 2002. doi: 10.1109/SFCS.2002.1181950.

[87] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6): 2551–2567, June 2006. ISSN 0018-9448. doi: 10.1109/TIT.2006.874390.

[88] H. Shin and J-S. Park. Optimizing random network coding for multimedia content distribution over smartphones. *Multimedia Tools and Applications*, 76(19):19379–19395, Oct 2017. ISSN 1573-7721. doi: 10.1007/s11042-015-3089-0. URL `https://doi.org/10.1007/s11042-015-3089-0`.

[89] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and T. Larsen. Network coding for mobile devices - systematic binary random rateless codes. In *2009 IEEE International Conference on Communications Workshops*, pages 1–6, June 2009. doi: 10.1109/ICCW.2009.5208076.

[90] D. S. Lun, N. Ratnakar, R. Koetter, M. Medard, and E. Ahmed. Achieving minimum-cost multicast: a decentralized approach based on network coding. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 1607–1617 vol. 3, March 2005. doi: 10.1109/INFCOM.2005.1498443.

[91] D. S. Lun, N. Ratnakar, M. Medard, R. Koetter, D. R. Karger, T. Ho, E. Ahmed, and Fang Zhao. Minimum-cost multicast over coded packet networks. *IEEE Transactions on Information Theory*, 52(6):2608–2623, June 2006. ISSN 0018-9448. doi: 10.1109/TIT.2006.874523.

[92] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. M. G. M. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 51(6):1973–1982, June 2005. ISSN 0018-9448. doi: 10.1109/TIT.2005.847712.

[93] C. Fragouli and E. Soljanin. Network coding fundamentals. *Foundations and Trends® in Networking*, 2, 01 2007. doi: 10.1561/1300000003.

[94] R. Ghrist and S. Krishnan. A topological max-flow-min-cut theorem. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 815–818, Dec 2013. doi: 10.1109/GlobalSIP.2013.6737016.

[95] Y. Wu, P. Chou, and S-Y.Kung.

[96] Y. Wu and S-Y. Kung. Distributed utility maximization for network coding based multicasting: a shortest path approach. *IEEE Journal on Selected Areas in Communications*, 24(8):1475–1488, Aug 2006. ISSN 0733-8716. doi: 10.1109/JSAC.2006.879356.

[97] C. Fragouli, J. Widmer, and J. . Le Boudec. A network coding approach to energy efficient broadcasting: From theory to practice. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–11, April 2006. doi: 10.1109/INFOCOM.2006.45.

[98] D. E. Lucani, M. Stojanovic, and M. Medard. Random linear network coding for time division duplexing: When to stop talking and start listening. In *IEEE INFOCOM 2009*, pages 1800–1808, April 2009. doi: 10.1109/INFCOM.2009.5062100.

[99] D. E. Lucani, M. Medard, and M. Stojanovic. On coding for delay—network coding for time-division duplexing. *IEEE Transactions on Information Theory*, 58(4):2330–2348, April 2012. ISSN 0018-9448. doi: 10.1109/TIT.2011.2177562.

[100] D. E. Lucani, M. Médard, and M. Stojanovic. Systematic network coding for time-division duplexing. In *2010 IEEE International Symposium on Information Theory*, pages 2403–2407, June 2010. doi: 10.1109/ISIT.2010.5513768.

[101] J. Krigslund, F. Fitzek, and M. V. Pedersen. On the combination of multi-layer source coding and network coding for wireless networks. In *2013 IEEE 18th International Workshop on Computer Aided Modeling and Design of Communication*

Links and Networks (CAMAD), pages 1–6, Sep. 2013. doi: 10.1109/CAMAD.2013.6708078.

[102] S. Sorour and S. Valaee. On minimizing broadcast completion delay for instantly decodable network coding. In *2010 IEEE International Conference on Communications*, pages 1–5, May 2010. doi: 10.1109/ICC.2010.5502758.

[103] L. Keller, E. Drinea, and C. Fragouli. Online broadcasting with network coding. In *2008 Fourth Workshop on Network Coding, Theory and Applications*, pages 1–6, Jan 2008. doi: 10.1109/NETCOD.2008.4476183.

[104] M. Toemoeskoezi, F. H. P. Fitzek, D. E. Lucani, M. V. Pedersen, and P. Seeling. On the delay characteristics for point-to-point links using random linear network coding with on-the-fly coding capabilities. In *European Wireless 2014; 20th European Wireless Conference*, pages 1–6, May 2014.

[105] M. Pedersen, J. Heide, and F. H. P. Fitzek. Kodo: An open and research oriented network coding library. In Vicente Casares-Giner, Pietro Manzoni, and Ana Pont, editors, *NETWORKING 2011 Workshops*, pages 145–152, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-23041-7.