ISCTE IUL

University Institute of Lisbon

Department of Information Science and Technology

Implementation of Linear Network Coding Over a Flexible Emulator

Nuno Bettencourt Coelho

A Dissertation presented in partial fulfilment of the Requirements for the Degree of

Master in Computer Engineering

Supervisor

Dr. Francisco António Taveira Branco Nunes Monteiro, Assistant Professor

ISCTE-IUL

Co-Supervisor

Dr. Rui Jorge Henrique Calado Lopes, Assistant Professor

ISCTE-IUL

December 2016

“Strive not to be a success, but rather to be of value.”

Albert Einstein

Abstract

This dissertation has the main objective of study and implement network coding (NC) techniques in a flexible emulator, programmed in a language that allows the coexistence of entities running parallel code, in order to emulate each node independently.

The dissertation starts with the study of NC's concept and with the characterization of the different type of coding methods, with a focus on linear network coding (LNC).

. A flexible Java emulator (named Net Genius) was developed, which not only allows numerous topologies of networks, but also different types of coding. In addition, the emulator allows to emulate the networks in two different modes: with a distributed network or with a centralized network. In order to present the differences between the LNC approach and the traditional approach used in packet networks (based in routing tables), the emulator allows the user to choose between these two types of approach, assessing the impact of having network coding over user-defined networks.

When implementing LNC, the concept of generations of packets was introduced in order to avoid combining packets from different sources. Leveraging on this, the transfer matrix at each node is calculated based on the coded packets and not based on the information stored in each node. In addition to this, a mechanism to code packets at the source was implemented, as well as a mechanism to introduce errors in the connection links. This allowed to emulate networks with different link error probabilities, in order to assess the resilience of the different approaches to the presence of failures.

Keywords

Linear Network Coding, Node Protection, Java Emulation, Software-Defined Networks.

Resumo

Esta dissertação visa estudar e a implementar técnicas de *network coding* (NC) num emulador flexível, programado numa linguagem que permita a coexistência de entidades a correr código em paralelo por forma a simular cada nó de forma independente.

Este trabalho começa com estudo do conceito de NC e da caracterização dos diferentes tipos de métodos de codificação, focando-nos essencialmente no *linear network coding* (LNC).

Optou-se por criar um emulador flexível em Java (designado por *Net Genius*), que não só permite várias topologias de redes, mas também vários tipos de codificação. Além disso, o emulador permite emular as redes em dois modos diferentes, um modo com uma rede distribuída e outro com uma rede centralizada. De modo a evidenciar as diferenças entre a abordagem LNC e a abordagem tradicional (sem codificação), o emulador permite escolher o tipo de abordagem em cada emulação, o que permite estudar o impacto do NC em redes definidas por utilizadores.

Procedeu-se à implementação de técnicas LNC e introduziu-se um conceito de gerações de pacotes, de modo a evitar a codificação de pacotes de diferentes fontes. A par disto, a matriz de codificação é calculada com base nos pacotes codificados e não com base na informação guardada em cada nó.

Por último, implementou-se um mecanismo para codificação de pacotes na fonte e um mecanismo de introdução de erros nos *links*, permitindo emular a rede com diferentes probabilidades de erro, sendo possível ver como as abordagens resistem à existência de falhas nas ligações.

Palavras-chave

Linear Network Coding, Proteção dos Nós, Emulação Java, Redes definidas por Software.

Acknowledgments

In first place, I would like to thank my supervisors Dr. Francisco António Taveira Branco Nunes Monteiro and Dr. Rui Jorge Henrique Calado Lopes for giving me the chance to work on such an interesting topic, for their support and motivation. I'm very thankful to my co-supervisor for his crucial programming advices.

I am especially very thankful to Dr. Francisco Monteiro for all the knowledge passed, for being always available, for the enormous patience, for all the hours spending in supervision, for his contribution, advices and crucial feedback when reviewing this work.

Secondly, I am thankful to Instituto de Telecomunicações (IT) for grating me a scholarship under the project L-DimM- NetCod funded by IT and Fundação para a Ciência e a Tecnologia, which allowed me to focus on this research.

I would like to thank my parents and sister that always help me and push me forward, giving me motivation to accomplish my objectives, not only on this work but also in my Master Degree. I am especially thankful to my parents, without their advices I would not be writing this words. Let me be at least half the persons my parents are. Thank you to my sister Mariana, for being the person you are and for always helping me along these years. I hope they are proud of me.

I would like to thank my girlfriend who was crucial during this work. Thank you for the enormous patience, for all the motivation and advices. Thank you for being in my life.

Finally, an acknowledgment to my friends. They know who they are. They help me along the last years. Thank you for all the help and advices. A special acknowledgment to Angela Borges for the unconditional support that proof that distance does not change anything.

Contents

List of Figures

Acronyms

CDF	Cumulative Distribution Function
CF	Compute-and-Forward
CRC	Cyclic Redundancy Check
DF	Decode-and-Forward
DS	Design Science
DSR	Design Science Research
GUI	Graphical User Interface
LNC	Linear Network Coding
MIMO	Multiple-Input-Multiple-Output
MVC	Model-View-Controller
NC	Network Coding
N-ML	Neighbour-Matrix Level
OSI	Open Systems Interconnection Model
PLNC	Physical-layer Network Coding
QoS	Quality of Service
RLNC	Random Linear Network Coding
SA	Slotted Aloha
SATCOM	Spectrum Efficiency of Satellites Communications
SBPP	Shared Backup Path Protection
SDN	Software-Defined Networking
SIC	Successive Interference Cancellation
S-IM	Source-Intermediate Matrix
WMN	Wireless Mesh Network

Symbols and Notation

Symbols

h_x	Transfer vector of packet *x*
$\mathbf{HL}_{\mathrm{V}}$	Transfer matrix at node *v*
$\mathbf{v}_{\boldsymbol{m}}$	Incoming transfer vector
P_e	Error probability
$\oplus$	XOR
$\mathbf{P}_{\mathrm{D}}$	Decoding probability
$\mathbf{C}$	Connection matrix
$\mathbf{G}$	Matrix received by a destination node
$\mathbf{H}$	Transfer matrix
$\mathbf{N}$	Noise matrix
$\mathbf{X}$	Packets matrix
$\mathbf{Y}$	Matrix received by a destination with noise
CRa	Acceptance criteria
L	Packet size
P_i	Packet
R	Receiver
g	Destination node
m	Matrix size
n	Integer number
r	Relay node
s	Source node
v	Node

x	One packet
k	Number between 0.0 and 1.0
$\bar{z}$	Average of *z*
z	Result of the emulation

Notation

$\mathbb{F}_q$	Finite field with cardinality q
$GF(2)$	Binary Galois Field
$\mathbf{H}^{-1}$	Moore-Penrose pseudo-inverse matrix
ρ	Error margin
τ	Auxiliary variable in the calculation of ρ

Chapter 1

Introduction

This chapter gives a brief overview of the work, presenting the objectives of this dissertation, as well as the research method. It also introduces the concept of Network Coding and provides a brief introduction to linear network coding over packet networks.

1.1 - Context

Computer networks have suffered huge several changes over the years. Traditionally, the routing of data in a network was pretty simple, since the intermediate nodes intervening in the routing of packets by simply forwarding the packets from a source node to a destination node, without processing them. Today, there is another approach that allows the intermediate nodes to process the packets between them in order to improve the network performance. This approach is called network coding (NC). Network coding is a method mainly used to optimise routing of data in digital networks, which was initially presented as a method for error correction [1]. In general, this method guarantees an efficient communication between the source and the destination. Not only ensures an efficient communication, but it also allows, in certain cases, recovering lost data. Traditionally, when a node wants to forward a packet to some other node, it simply forwards the packet, in order words, it simply repeats the packet. With network coding, a node has the capability of combining the n different packets it has received, in order to forward the new coded packet originated by the linear combination. Despite some limitations in certain wireless networks, network coding presents good performances in multihop wireless networks, such as wireless mesh networks.

One of creators of NC, Frank. R. Kschischang, argues that network nodes are capable of forming outgoing flows from various incoming flows by combining them not only time-multiplexing them, which means that the idea of network coding is to have intermediate nodes computing and transmitting functions of the packets that they receive.

Network coding research area is very important to both wired and wireless networks that have several applications in practical networking systems. This field of information theory uses linear combination of packets that requires some computational capabilities at the network nodes. Although it uses computational processing, the technology is cheap. NC is related with many areas, as shown in Figure 1.1.

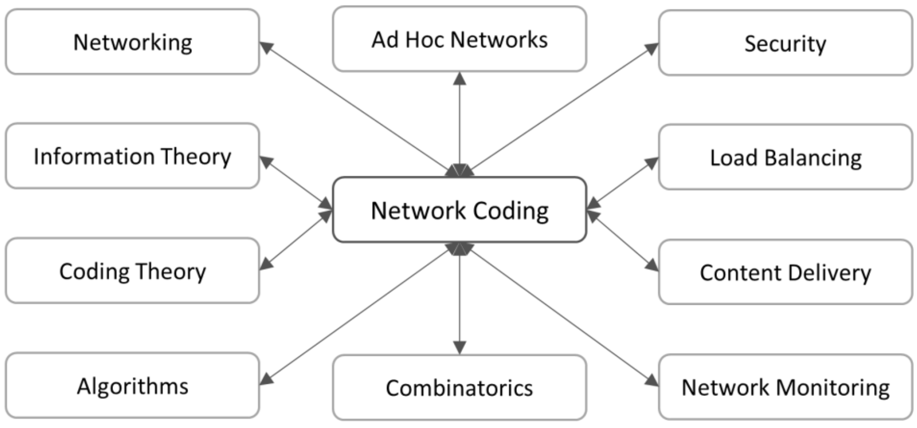


Figure 1.1 – The related areas of network coding.

Despite network coding having emerged around the turn of the millennium, there are not too many practical applications. However, the study of network coding allowed the academics Daniel Lucani and Frank Fitzek of the University of Aalborg, to create a Danish Start-up in 2014 with the goal of providing a reliable fast cloud storage technologies using NC. They have produced a simulator and a protocol for new generation networks based on network coding [2] [3]. In the recent years the European COST Action "IC1104 Random Network Coding and Designs over GF(q)" focused on this topic and several of its members produced proposals for network virtualisation based on network coding [4], [5], [6].

When using network coding the networks will have different behaviours. The study of these behaviours in several network topologies is important to understand the feasibility of using NC in wireless networks.

1.2 - Motivation

Until now, all telecommunications systems share the channels through time multiplexing, frequency or code. In the recent years, a new way of optimising the resources has been found. This new way, allows direct interference of signals from multiple users at the physical-layer level, forming signals which combines information from multiple users. With this, the spectral efficiency increases, reducing the channel use rate by a same amount in the data exchange between users. Combining information from multiple users provides a new mean way to tackle data errors.

Network coding is a way of combining this information that was firstly introduced to increase the flow of data across wired networks. Today, the goal is to find the best way of using network coding in both wired and wireless networks in order to provide both better performance and physical-layer secrecy. In general, this method guarantees two major points, i.e., an efficient and robust communication in case of link failures [7].

1.3 - Contribution

It is intended to create a flexible software program which will allow the analysis of the behaviour of network coded networks, programmed in a language that allows the coexistence of entities running parallel code. The results will be important to study the better linear network coding strategy and to evidence the differences between the network coding approach and the traditional approach.

This work deals with the packet transmission in both wired and wireless networks, while minimizing existing errors due to failures in network links, as well as ensuring the secure information exchange at the physical-layer, preferably without applying encryption.

1.4 - Research Questions

This work will be focused on finding the best way to use network coding in order to guarantee a good performance and ensure physical-layer secrecy. To that end, the research will be conducted around these two research questions:

1. Which packet coding and decoding techniques provide the best performance and simultaneously minimise network errors?

2. Which network coding approach attains the best results in terms of performance, resilience, secrecy, and network throughput?

1.5 - Objectives

In this work, it is intended to study digital transmission protocols for both wired and wireless channels with time and frequency overlap, based on network coding principles applied at the physical-layer of the Open Systems Interconnection Model (OSI). It is also intended to study and implement network coding techniques. To this end, a software program (artefact) will be created and the results generated will be analysed, in order to find the best way of using NC in wireless networks. This artefact will be essential to this objective, because it will give useful information that will help reach the goal of this dissertation.

To calibrate and validate the artefact, the results generated will be compared with some results available in the literature [8].

1.6 - Research Method and Document Structure

In this work we will use the Design Science Research (DSR) method [9] as the research method that was, initially, approached by Herbert Simon’s in [10]. Design Science (DS) is a theory-discover who seeks to create and prove theories by creating and evaluating new artefacts. For DS, an artefact is a major source of data, regardless the definition of artefact. The research model that will be used follows the next steps:

1. Problem identification and motivation;
2. Problem solution and objectives;
3. Design and build of the artefact (solution);
4. Evaluation;
5. Theorisation and justification;

1.7 - Communication

The first two steps are already completed. The third step is crucial to reach the objectives of this work, because the new artefact will help to understand and improve the behaviour of network coding in wireless networks by evaluating the results generated that will answer the research questions. Above all, this step leads to an artefact that solves the problem, if the produced artefact is well implemented. The evaluation will be through the analysis of the new techniques gains, in terms of the performance of the network packet successful transmission rate, delay and related metrics. This results will be compared with those obtained with traditional systems at the physical-layer. In order to contribute to the network coding research, steps 5 and 6 will complete this work.

This dissertation is organized as follows:

- Chapter 2 – will introduce network coding, in particular linear network on packet networks and how it can be implemented over the existing protocols. Also, it will provide an overview of the related work that is relevant for this work and for the network coding research area;
- Chapter 3 – will explain how the emulator was developed and the first steps of this work;
- Chapter 4 – will present the main ideas of this work and describe our major work as well as the results;
- Chapter 5 – will conclude this work by presenting the final conclusions and future work.

Chapter 2

Foundations of Linear Network Coding

This chapter provides an overview of the related work existing in the literature. It presents some of researches done on network coding since the beginning of the millennium.

2.1 - Introducing linear network coding

Linear network coding (LNC) is a coding method that maintains information about the data processing that is updated at each node that uses network coding [11]. With LNC the network throughput is increased, because the packet transmissions are more efficient [12], since information packets that didn't arrived to their destination can be achieved from linear combinations of other packets. Also, LNC brings more security and complexity to the network [13], as we saw in Chapter 1. LNC requires transmitting additional information with the messages and the operations on nodes are constrained to be linear of a finite field. This extra information is easily placed in a packet header, similar to the existing header in widely used packet networks and for that reason it is very easy to implement LNC over the existing protocols, like the IP protocol. This extra information is usually a vector of coefficients describing the processing. In a packet network the coding is made above the physical-layer, unlike in network information theory, in which the coding is made at the physical-layer. In LNC the symbols in the packet are taken from the finite field $\mathbb{F}_q$ and a packet is row vector over $\mathbb{F}_q$. Working over a finite field, the packet alphabet *B* is a space of row vectors over $\mathbb{F}_q$The smallest possible finite field one is *GF*(2) = $\mathbb{F}_2$, which only has two elements: 0 and 1. Therefore, in *GF*(2), is very easy to implement traditional operations, e.g., addition, division, multiplication and subtraction, and is very easy use linear algebra tools, e.g., matrices, Gauss-Jordan elimination, etc.

In LNC, local encoding functions are linear over $\mathbb{F}_q$. Since only linear operations are transformed in the network, each transmitted packet is a linear combination of the source node packets. We assume that each packet p_i is a scalar from $\mathbb{F}_q$. The received packets of source *s1* are columns vectors regarded as $\mathbf{X}_{I(s1)}$ with length *m* over $\mathbb{F}_q$. Assume $\mathbf{HL}_v$ as the transfer matrix at node *v*, a coefficient matrix from $\mathbb{F}_q$. Each column of the transmitted packets $\mathbf{X}_{\mathrm{R(v)}}$ of source node *s1* is related to $\mathbf{X}_{I(s1)}$ by the following linear model:

$$\mathbf{X}_{\boldsymbol{R}(v)} = \mathbf{HL}_v \mathbf{X}_{\boldsymbol{I}(v)} \tag{2.1}$$

Each row from $\mathbf{HL}_\mathrm{v}$, belonging to a determined edge $e \in R(v)$ is called a local encoding vector associated with *e*. Linear operations makes every packet transmitted along an certain edge a linear combination of the source node packets $p_1, p_2, \ldots, pr$. Note that at *v*, there is also a transfer matrix called $\mathbf{H}_v$, which is a matrix of coefficients from $\mathbb{F}_q$, given by the following linear model:

$$\mathbf{X}_{\boldsymbol{I}(v)} = \mathbf{H}_v \begin{bmatrix} \boldsymbol{p}_1 \\ \cdot \\ \cdot \\ \boldsymbol{p}_r \end{bmatrix} \tag{2.2}$$

Note that in order for a destination node g to be able to recover any packet p_1, $p_2, \ldots, p_r$ it is necessary to know the transfer matrix $\mathbf{H}_g$, such that the this transfer matrix has a left inverse that satisfies $\mathbf{H}_g^{-1}\,\mathbf{H}_g = \mathbf{I}_r$, assuming $\mathbf{I}_r$ as the $r \times r$ identity matrix. Note that after performing $\mathbf{H}_g^{-1}\,\mathbf{X}_{I(g)}$ at the destination node, g will produce $(p_1, p_2, \ldots, p_r)^T$. So, instead of taking pieces and decomposing in packets which later must be reconstructed in the right order, LNC combines into pieces such that any combination of these coded pieces allows to recover the original message, as long as we have a sufficient number of the combined pieces, as illustrated in Figure 2.1.

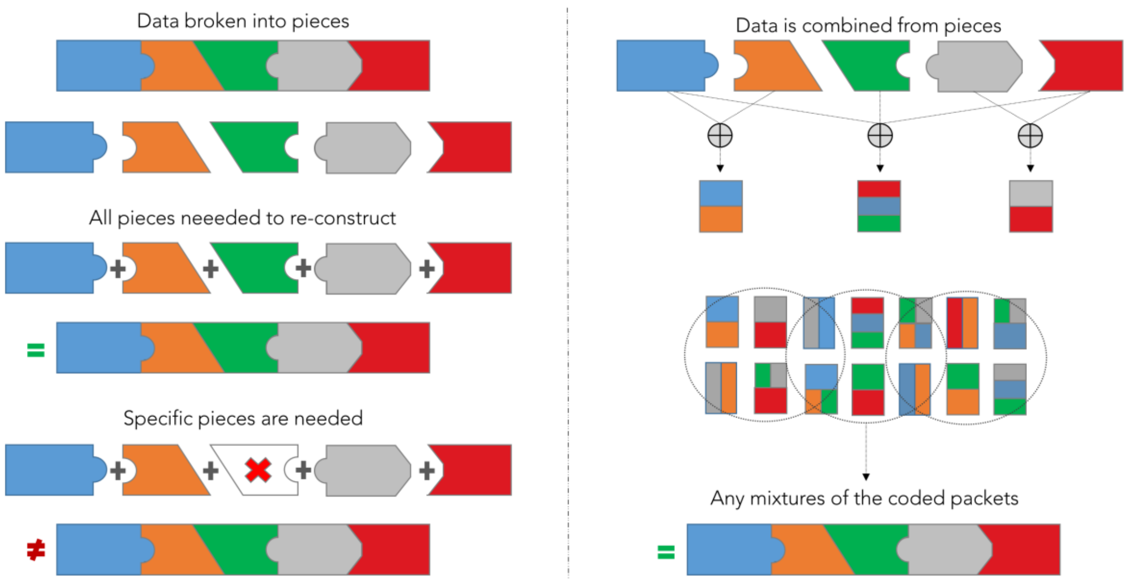

Data broken into pieces
All pieces neeeded to re-construct
=
Specific pieces are needed
≠
Data is combined from pieces
Any mixtures of the coded packets
=

Figure 2.1 - Concepts of RLNC versus the traditional approach in packet networks: construction of the coded packets and reconstruction of the original information at the receiver. [Inspired by a talk by Prof. Muriel Médard, MIT Research Laboratory of Electronics (RLE).]

In the example above, we see the difference between the traditional approach and the network coding approach, the left and the right approach respectively. The traditional approach does not allow coding, i.e., the network packets are always native packets. If a packet fails to reach the destination, the original data cannot be reconstructed, which means that the information is lost until the destination receives all needed packets. Besides losing information, in some cases we will need to re-send the lost packets which will have delays. With LNC, the packets are linearly coded over a finite field. One may say that these packets are versatile. Even if some packets fails to reach the destination, as long as the destination node receives a sufficient number of coded packets, those packets will suffice to recover the original data and there will be no need to re-send packets. Also, the LNC may be combined with a software-defined network (SDN) which allows to configure the coding method in the intermediate nodes. For networks will link failures or node failures, LNC will make the difference when compared with the traditional method, because it will help overcome the problem of network errors (bit error at the physical-layer level and packet losses at the network-layer level), given that any combination of packets is a good combination.

2.2 - Packet networks

Packet network are commonly used in linear network coding, mainly because, as said above, LNC requires transmitting extra information that can be accommodated in the packet header. In these networks, a message is composed by packets. Assume that the source message is composed by *r* packets. In LNC a packet transmitted at the node *v* is always a linear combination of one or more packets, which are the received packets at node *v*. In a finite field $\mathbb{F}_q$, each bits packet are grouped into vector of length m which are symbols taken from $\mathbb{F}_q$. Thus, each packet may be considered a vector of symbols of $\mathbb{F}_q$ [14].

Consider $p_1, p_2, \ldots, p_r$ as source packets (or source packet vectors) with n row length from $\mathbb{F}_q$. $\mathbf{X}$ is the $r \times n$ matrix whose i-th row is x_i. If we consider a destination node g, $\mathbf{Y}_g$ will be the matrix whose rows are the received packets at node g. In short both of the matrices are related by the following linear matrix model:

$$\mathbf{Y}_g = \mathbf{H}_t\mathbf{X} \tag{2.3}$$

Using this model, linear network coding allows the destination nodes to correctly decode the received packets, and also allows the intermediate nodes (intermediate nodes) to correctly encode the received packets. Note that the encoding may be done by the source nodes. Networks using LNC with *i* source nodes over $\mathbb{F}_q$, the coding vectors are in $\mathbb{F}_q^i$. The biggest concern is to ensure linear independence conditions to guarantee that the encoding and decoding at the network nodes are properly made. We may also apply LNC to the source by first coding at the source nodes. The sources may encode the packets before routing as the intermediate nodes does. This way we may increase the network's security and performance by increasing the protection against wiretapping attacks and by increasing the probability of the destination nodes to correctly decoding the information.

2.3 - Network Coding Over The Existing Protocols

As we explain, in packet networks the extra information that LNC requires is easily placed in a packet header, which makes network packets widely used for network coding methods [13] and very easy to implement over the existing protocols [15], like the TCP protocol. Implementing NC over existing protocols is not that hard, as shown by the authors in [16]. As known, TCP uses acknowledgments (ACK) for packet arrivals in correct sequence order in order to avoid congestion and ensure a reliable routing of data. When we use network coding, we need to change the TCP ACK mechanism. TCP always works with packets that are ordered. For TCP we need to address the TCP packet sequence number, because the coded packets may not be well ordered. In the destination node, when a packet is received from the source node, network coding is performed and a new TCP ACK number is generated, i.e., a destination node only acknowledges a packet when it receives a new linear combination of packets, even if after decoding the packets an information packet is not decoded, because the destination node may need more linear combinations to decode packets. This mechanism prevents retransmitting data and implement NC over TCP/IP.

2.4 - First approaches

Despite having been introduced for wired networks to increase the flow of data, network coding is not limited to that goal. Soon, researchers knew that NC could be explored in wireless networks not only as a method to increase the flow of data, but also as a method to minimize network errors and for error corrections.

In [1], the authors presented network coding as a method for error correction. Their work may be considered as network coding's first classic scientific article. They proved how this method allows one to transmit more information across a network. They defined network coding as a coding that takes place at the network nodes, which allows to transmit more information at the same time by combining information to be transmitted. This combination of information in wireless networks is only possible due to the direct interference of signals from multiple users at the physical-layer level, aforementioned. Due to this combination, NC can be used to tackle data errors, so the authors initially introduced NC as an error correction method. Their error correction approach is important and similar to the one that will be used in this work.

In the case of wireless networks, co-operation between nodes is crucial to get the benefits of spatial diversity, which is a way of taking advantages of the fading in wireless links. This benefit contributed to an extensive research on multiple-input-multiple-output (MIMO). NC may be classified into *intra-session* and *inter-session* network coding [17]. *Intra-session* scheme mixes the received packets within the same flow. The latter scheme mixes the packets from different flows (packets originated from different source nodes).

In [18], the authors showed how network coding can be useful in wireless networks, like wireless mesh networks. With NC, intermediate nodes may send combination of information, which brings benefits like potential throughput improvements and a network with a higher robustness degree. In this primer approach, they describe what is network coding and how it works, in terms of coding and decoding packets through linear combination. This NC combination is not a concatenation, which means that the linear combination of two packets with size L, results in a coded packet with also size L. To exemplify the benefits of NC method, they used the classic butterfly network (with link capacity 1), which is shown in Figure 2..

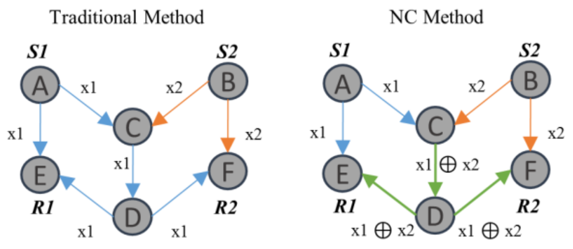


Figure 2.2 - Butterfly network where S1 and S2 multicast to both R1 and R2.

In their work, they stated that NC may have a major impact on the design of new information dissemination, e.g., in distributed storage systems [19] [20] and networking protocols.

The authors of [15] proposed a new architecture for wireless mesh networks. Basically, they used (not only in theory, but also in practice) linear combination to prove that mixing packets leads to an increase of the network throughput. To mix packets, they used the XOR (⊕) operation. These authors also presented network coding improves applied to the classical butterfly network (with link capacity 1) similar to Figure 2..

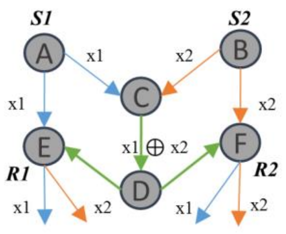


Figure 2.3 - Butterfly network showing the NC improvement throughput.

They derived the network coding gain for several network topologies. Their work was extremely significant, showing the packet coding algorithm and a flow chart for their wireless mesh network (WMN) architecture. In their packet coding algorithm they presented the probability of a node decoding a native packet (original packet). A native packet is a packet that is not coded, i.e., is a traditional packet. Suppose the node encodes n packets by linearly combination. Let P_i be the probability that a next hop r_1 has heard the coded packet i. The probability, P_D, of r_1 could decode its native packet is equal to the probability of r_1 hearing all $n-1$ native packets combined with its own, as follows:

$$P_D = P_1 \times P_2 \times \ldots \times P_{n-1} \tag{2.4}$$

They used this probability to estimate if a neighbouring node would be capable to decode its native packets. They presented good results in wireless mesh networks and their ideas can be explored in other types of wireless networks.

2.5 - Protection against node failures

The authors in [8] focused on using network coding as the basis of a novel protection scheme in multihop wireless networks. The protection scheme was tested with one and two intermediate nodes failure. They analysed the network behaviour by measuring the quality of service (QoS), i.e., jitter, latency and the packet delivery ratio. In their work, they used a wireless mesh network, as depicted in *Figure 2.*. A WMN may be based on the IEEE 802.11s standard. These networks are capable of establishing several wireless links between client nodes and multiple destinations, and its network structure looks like a mesh. The structure aggregates sources (clients), router nodes (intermediate nodes) and one or more destinations. The destination provides Internet access to the clients and to the intermediate nodes. The intermediate nodes are responsible for making the forwarding decisions based on their knowledge of the network. The intermediate nodes combine packets from different sources increasing reliability. However a wireless network is exposed to many types of interference. In addition to the interferences in WMNs, the intermediate nodes may also suffer from random failures. NC is a way of protecting from intermediate nodes failures.

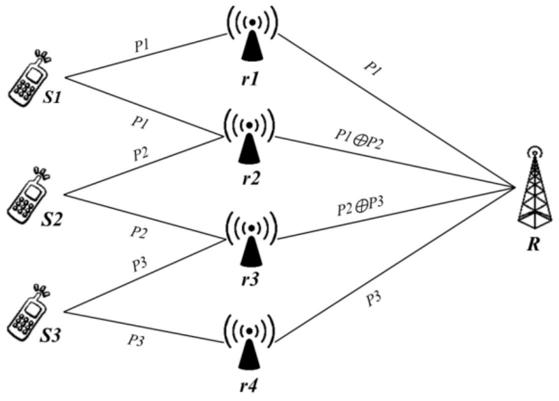


Figure 2.4 - Network model for protection against the failure of one intermediate node.

The network model in Figure 2. has three sources (S_1, S_2 and S_3), four intermediate nodes (r_1, r_2, r_3 e r_4) and one receiver (R). As shown, the sources can only communicate with the intermediate nodes, to which they are connected, due to antenna coverage limitations. Each intermediate node is capable of coding packets from different sources, using the XOR coding technique, and the receiver R is able to decode these packets from different intermediate nodes.

The XOR protection scheme is very simple to use. Assume that the sources are sending packets to the destination through the intermediate nodes and assume that sources are independent from each other. Let $P_1 = \{0, 0, 1, 1\}$ and $P_2 = \{1, 0, 1, 0\}$ be packets from S_1 and S_2 respectively. Let $P_1 \oplus P_2$ be the coded packet generated by r_2 from packets P_1 and P_2. This intermediate node will forward the new packet to R. Intermediate node r_2 applies the XOR operation to the received packets and r_2 generates this new packet as follows:

$$P_1 = \{0, 0, 1, 1\}$$
$$P_2 = \{1, 0, 1, 0\}$$
$$P_1 P_2 = \{1, 0, 0, 1\}$$

In this case, the receiver node can retrieve the information supposedly lost for a single intermediate node failure, because all sources are connected to at least two intermediate nodes. Intermediate nodes r_1 and r_4 will forward packets P_1 and P_3. Intermediate nodes r_2 and r_3 will code the packets $P_1 \oplus P_2$ and $P_2 \oplus P_3$ respectively.

If r_1 fails to forward packets from the source to the destination, R can still obtain all packets. P_3 is obtained from r_4, then it decodes P_2 from r_3 and r_4. Then it decodes P_1 from r_2, r_3 and r_4. We may affirm that if only one intermediate node fails to forward packets from the sources to the destination, R still obtains all packets. Therefore there is no information lost if *only one* intermediate node fails or if only one intermediate node link fails.

Also they used another network model, as shown in Figure 2., for the protection against the failure of two intermediate nodes.

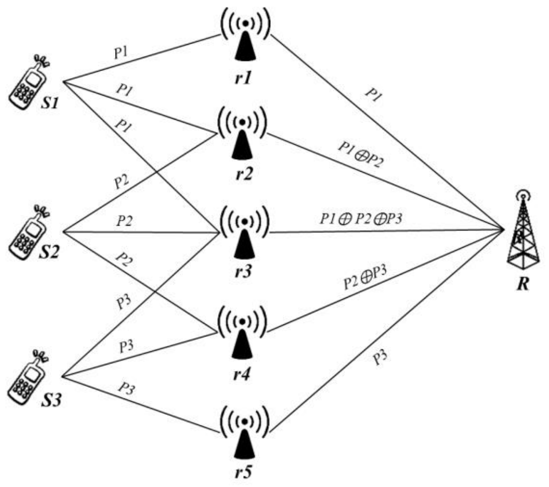


Figure 2.5 - Network Model for protection against the failure of two intermediate nodes.

In this network model, intermediate nodes r_1 and r_5 will forward the packets P_1 and P_3 respectively. Intermediate nodes r_2 and r_4 will code the packets $P_1 \oplus P_2$ and $P_2 \oplus P_3$ respectively. Finally, the intermediate r_3 will code the packet $P_1 \oplus P_2 \oplus P_3$.

If intermediate r_1 fails to forward packets from the source to the destination, R still obtains all packets because P_3 is obtained from r_5, then it decodes P_2 from r_4 and r_5. Then it decodes P_1 from r_2, r_4 and r_5. If intermediate r_1 and r_2 fails to forward packets from the sources to the destination, R still obtains all packets. P_3 is directly obtain from r_5 and it decodes P_2 from r_4 and r_5. Then, P_1 is achieved by decoding the packet from r_3 with the packet from r_4. R can decode all information from the intermediate nodes if at most two of them fails to forward. Therefore, the network is protected against one or two node/link failures, but only if all sources are connected to at least three intermediate nodes.

An intermediate node that uses the XOR scheme, must code all packets from the sources to which it is connect in order to send the new coded packets to the destination. Assume the network model scenario, presented in Figure 2.. Let P_1 and P_{11} be packets from source S_1 and P_2 and P_{22} be packets from source S_2. Both P_1 and P_2 were send from their sources before than P_{11} and P_{22} respectively. Intermediate node *r2* must code P_1 with P_2 and P_{11} with P_{22}. A bad coding example would be $P_{11} \oplus P_2$ because if r_1 fails to forward P_1, this packet couldn't be retrieved by the destination.

2.6 - Protection against link failures and errors

In [21], the authors proposed a network coding protection scheme against errors and failures in network links. They used bidirectional unicast networks, like the one in Figure 2.. As known, network errors are common and typically may refer to alterations of the transmitted data unit, such that the network nodes do not know the existence of errors prior to decoding. The authors affirmed that errors in networks may not be handle with classical error control codes [22]. Their results shown that when adversary errors (errors in the packet) are introduced, for n_e paths with errors, $4n_e$ protections paths are sufficient for recovering the information at the end nodes. The results in [22] show that, when n_e errors and n_f failures (packet losses) occurs on the primary or protection paths, $4n_e + 2n_f$ are sufficient to correctly decode the information at the end nodes.

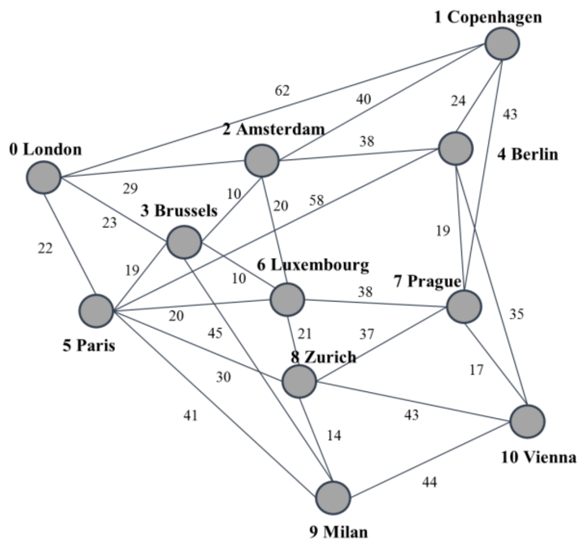


Figure 2.6 - COST239 network with 11 nodes and 26 edges in Europe.

In [23], the authors used NC to introduce a protection scheme against single and multiple link failures, recovering a second copy of each data unit transmitted without re-routing data or without failure detection. As mentioned before, in [8], the authors introduced a protection scheme against node failures. In [23] these authors' effort focused on a protection scheme against link failures. They apply the same strategy ensuring that in a single connection each node receives two copies of the same packet, in which one of the copies can be extracted from linear equations of packets. To do this end, they use the network coding technique. Their strategy ensures packets recovery without re-routing data or detecting failures, which means that the protection cost is not higher. In fact, is not too much higher than shared backup path protection (SBPP) strategies [24] while it still provides a faster recovery than SBPP.

2.7 - Communication protocol

In [25], the authors proposed a general method for designing the transmission protocol with binary physical-layer network coding (PLNC) where the communication protocol is specified by a binary matrix. They examined several proposed protocols in terms of energy consumption, error rate, and throughput performance, and decoding strategy. In their work, they considered a different network where each user's message is intended to all other users of the network (broadcast, multicast). They proposed a general method for a *K*-way intermediate network (tree based). Each network node has a degree less than *W* (specified energy constraint in terms of the maximum number of transmissions per user), where the construction involves the following steps:

1. Select an arbitrary node as the root;

2. Connect arbitrarily the remaining W nodes in the layer 1 to the root;

3. Connect arbitrarily the remaining $W - 1$ nodes in the layer 2 to each layer 1 nodes;

4. Repeat the previous step until all K nodes are included in the tree.

They use a binary matrix to specify the communication protocol. Naturally, any different structure of the binary matrix leads to a different coding and decoding strategy.

Many authors have researched on models to specify the communication protocol of the networks. Most of the researches, like the authors in [26], use the model defined as:

$$\mathbf{Y} = \mathbf{HX} + \mathbf{N}\,. \tag{2.5}$$

There exist some variations on this model. Regardless the model variations tough, **H** is the transfer matrix that defines the connection between the network nodes. **N** is noise matrix and **X** is a matrix to be estimated, which contains transmitted information. **N** corresponds to the thermal noise in communication links, distortion, interference, or problems within the network (like intermediate nodes failures). In this model, **Y** is also a matrix. It contains the information received by the destination. Usually this information is a linear combination, i.e. coded information transmitted by nodes.

Based on this model, some codes have been proposed by different authors. As shown in Figure 2., **H** (the coding matrix which replaces rows in some transfer matrices) must be a linear combination of the incoming encoding matrices ($v_{1},\ldots,v_{m}$) in order to guarantee that **H** remains invertible for each network destination (R) and maintains the full rank [11]. One of the purposes of these codes, investigated by Kötter and Kschischang, is to include in the packet the encoding matrices to provide the receivers with the accurate information to correctly decode the packets.

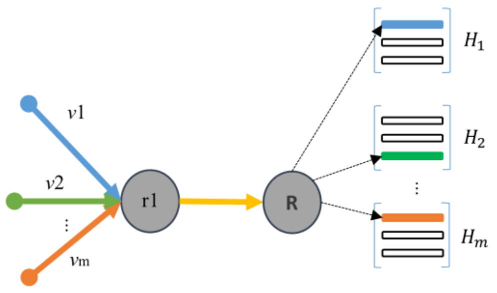


Figure 2.7 - Information flow algorithm example.

As shown in the Figure 2., the communication protocol and the protection scheme are very important. The right implementation of network codes allows the receiver (R) to correctly decode the packets sent by the sources (S_1, S_2, S_3) that were coded by the network intermediate nodes (r_1,

r_2, r_3 and r_4) and simultaneously allows to have the knowledge of the packet's route, with the information stored in the packet header.

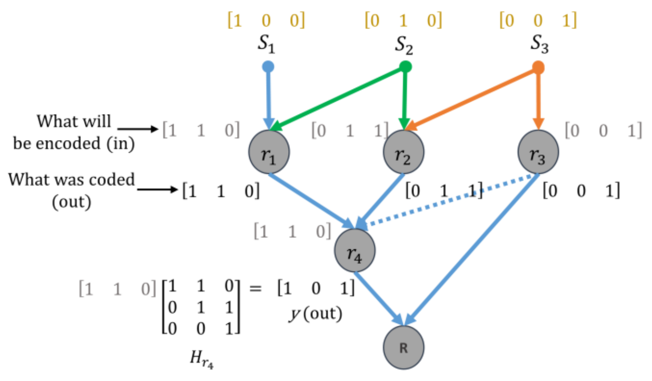


Figure 2.8 - Information flow scheme example.

2.8 - Other approaches

As already mentioned, node co-operation is very important when using network coding. Sometimes, it is beneficial to have some users acting like intermediate nodes in order to help other users recovering their information messages. This strategy is usually called as *physical-layer cooperation* which often includes schemes as *compute-and-forward* (CF) or *decode-and-forward* (DF) [27].

The system model proposed in [28] presented some interesting results. In their system model, as shown in Figure 2., intermediate nodes do not decode the information from the received packets but simply forward the coded packets. They consider a network with one source, one destination (R) and n intermediate nodes where all nodes are equipped with a single antenna. Binary random linear fountain code is used by the sources to code the packets. The coded packets are transmitted to the intermediate nodes and then to R. At the same time, some intermediate nodes are listening to the sources while others intermediate nodes are transmitting to the destination. A simple cyclic redundancy check (CRC) is performed on the received packet P by the intermediate nodes that are listening. A listening intermediate that recovers P, stores this packet in its buffer, if it can hold it. If CRC decoding cannot decode P successfully, then P is dropped. On the other hand, transmitting intermediate nodes randomly choose packets from their memory and forward them to R. Inter-intermediate interference is prevented because intermediate nodes use separated directional antennas. Successive interference cancellation (SIC) is used at the receiver to recover the coded packets sent by the intermediate nodes. SIC is a physical-layer capability that allows a receiver to decode packets that arrive in the same time instant. When the sources transmitted K

coded packets in an equal number of time steps, R starts to check if it is able to decode the received packets. If R decodes the packets after $N \geq K$ time steps, it sends a signal to the source in order to finish the transmission. Also in their model, they used a Markov Chain (random stochastic process with memoryless property where the present state does not depend on the previews states) to estimate the performance of their model by computing the erasure probability of the packet. The model presented a good erasure probability by selecting properly values for the memory size of each intermediate, the probability of listening and the number of intermediate nodes.

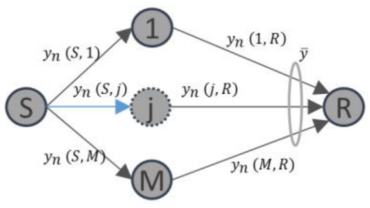


Figure 2.9 - Wireless network with one source, one receiver and *n* intermediate nodes.

The authors of [29], presented a novel to random access based on network coding at the physical-layer level, also known as *compute-and-forward* that was proposed in [27]. Basically, CF allows the intermediate nodes to decode linear equations of the information transmitted using linear combinations with noise, provided by the network channel. This strategy also provides protections against noise and guarantees that the receiver is capable of getting all messages by solving the linear equations, if it receives sufficient linear combinations. In their approach, when packets collide, the receiver tries to decode a linear combination of those packets using the CF approach instead of trying to decode the packets. They showed better results than other approaches in terms of throughput. By doing this, the receiver may recover all original packets. The difference between this approach and slotted ALOHA (SA) approach is that, in the latter, packet collisions leads to packet losses.

In [30] the authors also studied the SA collision problem. SA was developed under a collision model where waste is inevitable, given that some slots will always contain collided packets. They affirmed that random access should be used whenever there is an uncertainty about the network users that want to transmit at the same instant. Using successive interference cancellation allows to unlock the collisions slots. As seen before, some efforts combine SA with PLNC. The authors want to solve this issue with SIC across multiple slots. In this approach, nodes transmit copies of the same native packet in multiple slots and SIC is used on the receivers to remove the copies of the already recovered packets from the collisions slots. The recovering and removal of the packet copies is as shown in Figure 2., done in a successively manner.

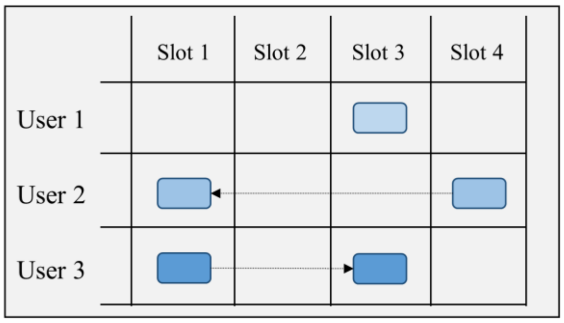


Figure 2.10 - Successive interference cancellation in SA.

In the Figure 2., the packet from user 2 is recovered in slot 4, which enables recovering the packet from user 3 in slot 1 by subtracting the copy of user 2 packet in slot 1. Since the packet from user 3 was recovered, it enables the removal of its own copy from slot 3, which allows the recovery of user's 1 packet. The author stated that in this example, using SIC guarantees a 0.75 packet/slot throughput, instead of 0.25 packet/slot throughput if weren't used SIC.

Slotted ALOHA is present in most of existing wireless random access protocols, albeit its inefficiencies. SA is very important, since it has been used during the last four decades, but the authors in [30] affirmed that coded random access (codes on graphs) provides new grounds to design communications systems, important no *machine-to-machine* devices.

Security is also a very important issue in the NC literature. The concept of secure NC was introduced in [31]. NC allows to send linear combinations of packets instead of native packets (uncoded data) which provides advantages of multipath diversity for security, like protection against wiretapping attacks. The systems that requires this type of protection can get it automatically with network coding, without implementing other security mechanisms. Figure 2. shows how network coding offers a natural protection against wiretapping attacks. Assume that node A wants to send a packet to node D through ABD and ACD. Assume also that an adversary Eve is capable of wiretap one single path. Without network coding, Eve can intercept the independent packets x_1 or x_2. When using network coding, packets x_1 or x_2 are coded and sent through the different paths, thus Eve cannot decode any part of the information. If Eve intercepts $x_1 + x_2$, the probability is approximately 50% of correctly decoding x_1, which is the same of random guessing [13].

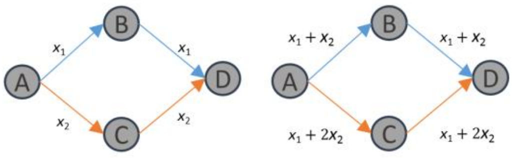


Figure 2.11 - Network coding natural protection against wiretapping.

Network coding can improve the spectrum efficiency of satellite communications (SATCOM). However, confidentiality is an important issue to be studied. In [32], the authors proved that NC can improve the system throughput, but it also improves the sum secrecy rate of bidirectional SATCOM. Their network coding scheme outperforms the traditional schemes that do not use NC. They incorporated the XOR NC scheme into SATCOM using NC, guaranteeing a secure bidirectional information transmission.

In [33], the authors proposed a model that minimizes the probability of interception by optimising the number of transmitted coded packets. In their model, the transmitters use random linear network coding (RLNC) to broadcast the information. They reported that there is no need for the receiver have knowledge of an adversary. Their work is also based on [34], where the authors affirmed that an arbitrary small intercept probability can be achieved by increasing the number of segments (source packets) of a source's message, although it increases the delay. To optimise their model, they aim to determine the optimal number of coded packet transmissions that minimizes the probability of interception. One way to reduce the interception probability is to measure the quality of the channel between the sender and the receiver. Let Alice be the sender and Bob the receiver, as it is traditional. If Alice measures the instantaneous quality of the channel between her and Bob and transmits the coded packets to Bob, only and only, when the channel has a good quality, the interception probability will decrease but the delay will increase, as expected. Their results showed how their resource allocation model can reach good results, optimising both delay and interception probability constrains.

All these efforts presented relevant proposals, with promising results. NC has been studied and supported in several international project, e.g., COST. We aim at contributing along these lines of research in order to improve network coding methods for reliable networks.

Chapter 3
The Emulator:
Net Genius

This chapter presents Net Genius, the emulator developed to emulate a network using NC. It describes all the important features of the emulator and the emulation environment. It also presents the development progress of the emulator.

3.1 - Network Emulation

In order to understand and improve the performance of network coding in computer networks we developed an emulator in software. Implementing NC in a software will be helpful to compare emulation results with theoretical results already known in literature.

Creating our own emulator give us a greater understanding of the advantages and problems of using network coding and allows us to test several scenarios and multiple network coding algorithms. It also give us the opportunity to replicate a network, leading us to a more detailed grain.

The emulator developed in this work was written in Java language. To run the emulator, the user has to open the emulator executable file, called “Net Genius 4.3.jar”. It emulates the routing of data via multiple threads. For example, each network node is a thread. Each thread is independent which makes the emulation run quicker and closer to reality. In order to develop this emulator, we needed to understand a network typical flow and the behaviour of network nodes so that we could properly emulate a network. The emulator needs to replicate the network to a certain level of detail. In this case we will not need a perfect emulator. A perfect emulator would perfectly emulate a real network. We need a tool that is based on theoretical parameters and respects the typical network behaviour. Understanding the network’s theoretical behaviour allows us to build the emulator closer to reality. The development process of an emulator takes several stages towards the release candidate (potential final version of a software) and release (final version of the software) versions. It is very important to decompose the project into smaller parts. The emulator will be a group of individual components that only have meaning when they are properly connected. In our work, we don’t intend to build an emulator that will interact with hardware, i.e., our emulations will be based on theory and applied only on software.

The project was divided in five main packages, as depicted in Figure 3.1: configuration, emulation, utils, view and genius. The configuration package contains the main class, which contains the main method and is the entry point of the emulator application. This class is from where the applications all start. The emulation package is the biggest package in terms of number of classes and sub packages. It contains the most important Java classes, like the emulation objects and resources, e.g., source, destination, bucket and packet. This package will contain the network objects package which will contain the aforementioned emulation objects and will also contain the resources package which will contain the aforementioned emulation resources. The utils package will contain some personalized data structure, operations and libraries. The view package will contain the user interface Java classes that will be later explained in this work. The genius package is only used in the emulator advanced mode, as it will be later explained. The user may read some documentation about the java classes through the doc/index.htlm file inside the project folder.

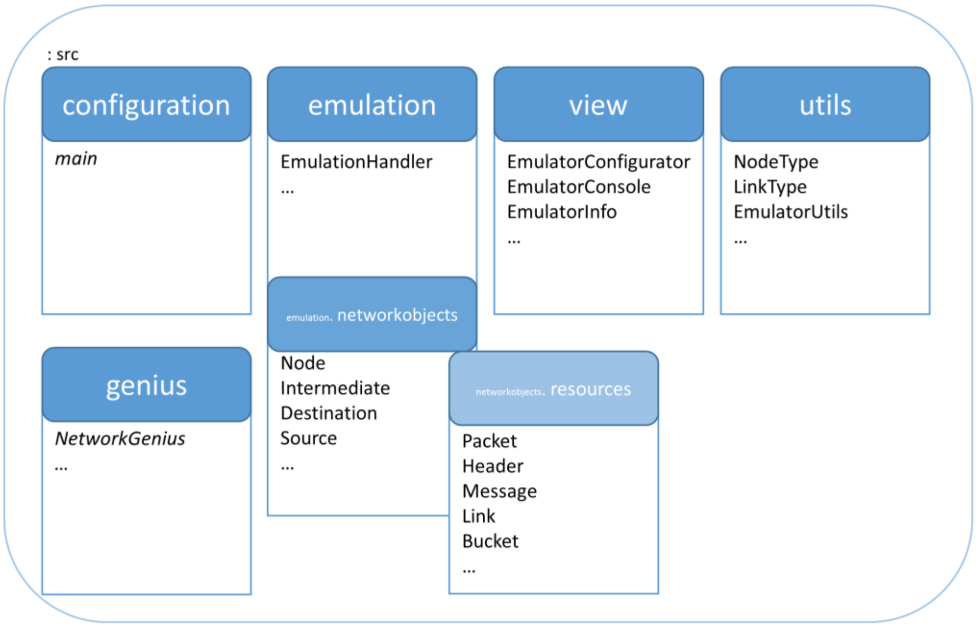


Figure 3.1 - Emulator project main packages.

With this project division we were able to implement the Model-View-Controller (MVC) software architecture pattern, as shown in Figure 3.2. Basically, MVC divides the project into three interconnected parts. In this work, the controller is responsible for connecting the model (objects, data, etc.) to the view (user interface, etc.). For example, when a destination object needs to send an exception to the user interface, it will be doing it through the EmulationHandler. Implementing MVC pattern makes the project easier to adapt in future situations. For example, if we want to change the source behaviour, we just have to change the source class in Model because it will not affect the rest of the project.

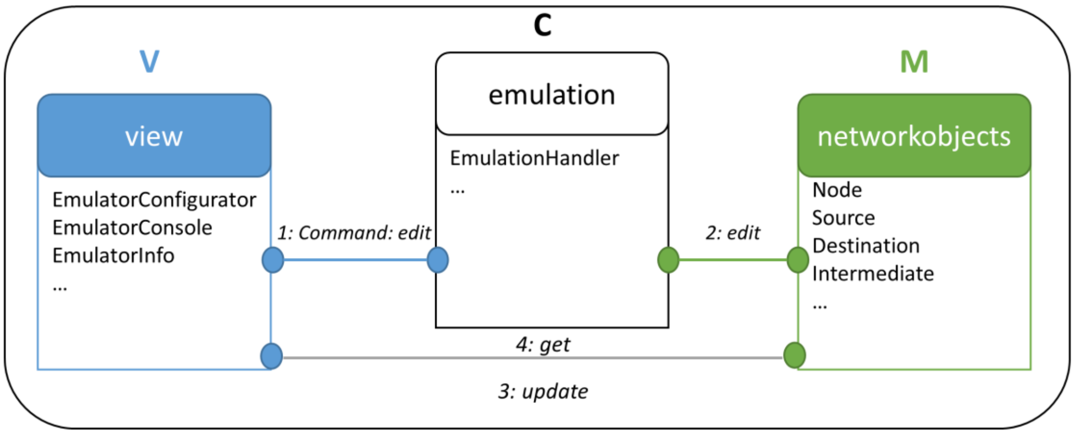


Figure 3.2 - MVC software architecture pattern.

3.2 - Type of Nodes

After having described the organisation of the emulation system in the previous section, the emulation components and environment will be now addressed. The emulation environment is composed of three types of network nodes, as seen in Figure 3.3.

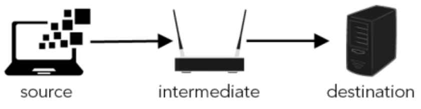


Figure 3.3 - Network node types.

The source node plays the role of the typical user, who wants to send several messages to “the internet”. To send those messages the source node typically cannot send the packets directly to the destination, which is a situation that happens quite frequently in upload scenarios. In some cases, the upload rate is lower than the download rate because the source nodes are not capable of sending the packets directly to the destination, since they have to send through several intermediate nodes. The download rate is typically higher than the upload rate because the destination is capable of sending the information directly to the user. Thus, the source nodes will be responsible for generating and sending messages to a certain destination in the network. To encode the packets, the source uses linear independent vectors with *linear lifting* [35], a typical approach encountered in linear coding. For example, if the source *s* wants to send packets p_1, p_2 and p_3 over $GF(2)$, the packets are seen as rows of **H** at the source.

Assuming that a certain number of packets belong to the same burst, the combination matrix $\mathbf{H}_S$ is defined as follows:

$$\mathbf{H}_s = \left(\begin{array}{ccc|c} 1 & 0 & 0 & p_1 \\ 0 & 1 & 0 & p_2 \\ 0 & 0 & 1 & p_3 \end{array}\right).$$

The intermediate node does *not* play the typical role it has in traditional packet networks. In traditional packet networks, the intermediate node role is to forward the received packets to the destination. In a NC context, the intermediate node firstly encodes the received packets and later forwards the coded packets to the destination. In our emulation, the intermediate node will be responsible for both encoding the correct packets and forwarding them to the destination. The destination node will be responsible for receiving and correctly decoding the received packets. In order to encode and decode packets the emulator uses a linear model that will be explained later in this chapter.

Figure 3.3 also displays links between nodes. These links are very important objects in the emulation environment chiefly because they impose a limit to the number of packets that can be transmitted during a time slot.

Given that the three types of nodes share some characteristics (name, ID, state, etc.), we generalise the concept as illustrated in Figure 3.4. Using the concept of inheritance, the information of each type of node is made manageable in hierarchical order.

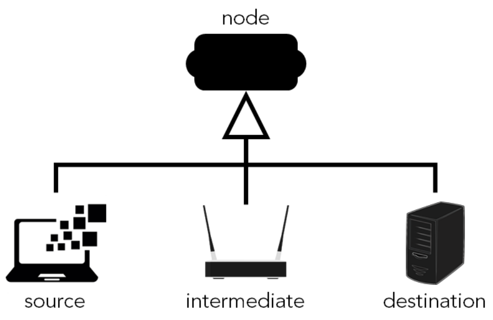


Figure 3.4 - Network nodes scheme.

Each node will have buffers that are responsible for storing both the received packets and the newly combined packets (also named *computed packets*) according to their connections. A computed packet is a packet that has already been processed and is ready to be sent. Let one consider the example where an intermediate node is connected to two sources and to three other intermediate nodes. This intermediate node will have two IN buffers and three OUT buffers, as it is shown in Figure 3.5. The difference between an IN buffer and an OUT buffer is that the first will store received packets and the second type will store computed packets.

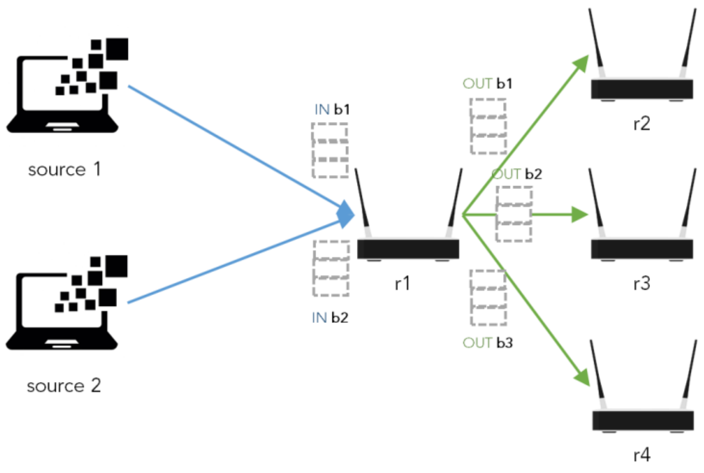


Figure 3.5 – The IN and OUT buffers at an intermediate node r_1.

3.3 - Packet Structure

The packet structure encompasses several information fields: some fields contain control information and others contain the message to be transmitted (payload), i.e., each packet contains a header and the message. The header stores control information, which is basically the packet's definitions needed for coding and decoding. The message comprises of one field, which is the packet data. Therefore, each packet will have 8 fields, as described in Figure 3.6.

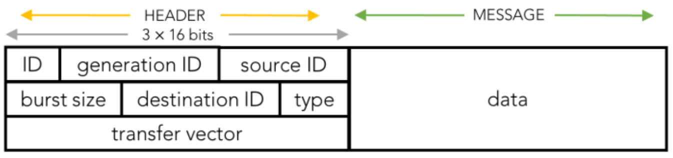


Figure 3.6 - Packet structure.

In the first versions of the emulator, some of these fields were not considered but in further versions the need for those fields came about, for example, the need to have a field containing a *generation ID* of the packet. In the header, the ID field identifies the packet. The ID of the packet is composed of the node ID and a sequential number. For example, when a packet is produced at the source node "s_1", his ID will be "s_1x" where x is the number of packets generated by that source node.

The *source ID* and *destination ID* fields respectively contain the source identifier and the destination identifier. These fields are important so that network nodes may correctly encode and decode the packets and forwarding the packets.

The generation ID identifies the generation number of the packet and this generation number is associated with the length of the burst (*burst size*), which is in this work named the "generation size". For example, if a burst has size four; there will be four packets belonging to the same generation.

Only with two values can be assigned to the field: *[INFORMATION, PROBING]*. These values indicate if a packet contains information or if it is a packet that is transmitted to just validate the transmission.

The *transfer vector* (or *coding vector*) will be used to encode and decode the packet itself and the data transmitted. In the decoding process, the transfer vector will be crucial to know the information of the packet. The size of the transfer vector is determined by the burst. In the message, the information field will contain a message or part of a message transmitted by a network node.

3.4 - Network Model

As it was described in Chapter 1, the authors in [8] proposed network coding as the basis of a novel protection scheme in multihop wireless networks. Their protection scheme was tested with one and two intermediate nodes failure. The paper analysed the network behaviour by measuring the quality of service (QoS). The first version of our emulator was based in the work of these authors.

Regardless the network topology, the information that the destination receives is defined by the following linear model:

$$\mathbf{Y} = (\mathbf{H} + \mathbf{\Sigma})\mathbf{X} + \mathbf{N}. \tag{3.1}$$

Matrix **H** is the transfer matrix that defines what will be coded between intermediate nodes and sources nodes, and $\mathbf{\Sigma}$ and **N** are both noise matrices. In the first versions of the emulator, **H** is always a global binary matrix (i.e., with values in $\mathbb{F}_2$ representing the whole network between the sources and a single final destination). Binary matrix $\mathbf{\Sigma}$ accounts for the imperfections in the overall transfer matrices (i.e., cuts in communication links) and the binary matrix **N** corresponds to the thermal noise in communication links, distortion, interference, or problems within the network (like intermediate nodes failures), depending of the level, i.e., physical-layer level or network level. In these versions, both perturbation matrices will be ignored. **X** is a matrix to be estimated, which contains the source's information packets. An example, the network model presented in Figure 2., has a matrix **H** defined as follows:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

The columns of **H** matrix represents the input links and the lines represents the output links. In **H**, '1' represents a connection from an intermediate node to an input. As r_1 is only connected to S_1 through the first input link, the first line of **H** is [1 0 0].

It is also useful to consider the binary matrix **G=Y,** defined as **G**=**HX**, which represents the packets sent from the intermediate nodes to the destination *R* without any node failures. Let us assume that $\mathbf{X}_1$ is a binary matrix with its rows representing the packets P_1, P_2 and P_3 sent by S_1, S_2 and S_3 respectively, and defined as

$$\mathbf{X_1} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Consequently, matrix **G** represents the packets P_1, $P_1 \oplus P2$, $P_2 \oplus P_3$, and P_3 sent by r_1, r_2, r_3 and r_4 respectively, and therefore it becomes

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

When the emulator starts, all sources begin to create information packets. The packet's content has equiprobable '0' and '1' values, i.e., they are created by sampling a uniform distribution with probability ½ for each symbol. The source's output capacity is directly related with the source's data rate and the packet size. When a packet is produced, the source sends it to the intermediate node to which is connected. Because the emulator is a multiple thread system, the intermediate nodes are always active and waiting for new packets from the sources.

An intermediate node that has only one IN buffer (one entry connection), automatically forwards the arrived packets to the destination through the destination link, if the link is available. Link availability depends on is capacity (i.e., the link capacity, in the information-theoretic sense). On the other hand, an intermediate node that uses the XOR scheme, must code all packets from the sources that are connected to it in order to send the new coded packets to the destination.

In the first versions of the emulator (i.e., where the focus was on the replication of the results in [8]), intermediate nodes coded packets from different sources. Assume the network model, presented in Figure 2.. Let P_1 and P_{11} be packets from source S_1 and P_2 and P_{22} be packets from source S_2. Both P_1 and P_2 were send from their sources before than P_{11} and P_{22} respectively. Intermediate node r_2 must code P_1 with P_2 and P_{11} with P_{22}. A bad coding choice would be $P_{11} \oplus P_2$ because if r_1 fails to forward P_1, this packet couldn't be retrieved by the destination. If an intermediate node is unable to combine the correct packets after a certain number of attempts, the intermediate node will simply forward the packet without coding. For example, in the case just presented, if r_2 never receives P_2 , after some time it will simply forward P_2 without coding the packet. The destination must decode the coded packets received also using an XOR scheme.

The way the destination decodes the packets change along these first versions of the emulator. In the earlier versions, the decoding algorithm is based on a neighbour-matrix level (N-ML) approach. To decode a packet from intermediate x, the destination needs all the other packets from the other intermediate nodes. This N-ML approach is based on an ordered (by coding level) matrix. Assume the network model presented in Figure 2.. Let $P_{1i,}$ $(P_1 \oplus P_2)_{i,}$ $(P_2 \oplus P_3)_{i,}$ $P_{3i,}$ be packets received by the destination by r_1, r_2, r_3 and r_4 respectively. Let *i* be the packets timestamp given by their sources when they are produced. To decode the packets, *R* must put them all in a matrix (***G***), as follows.

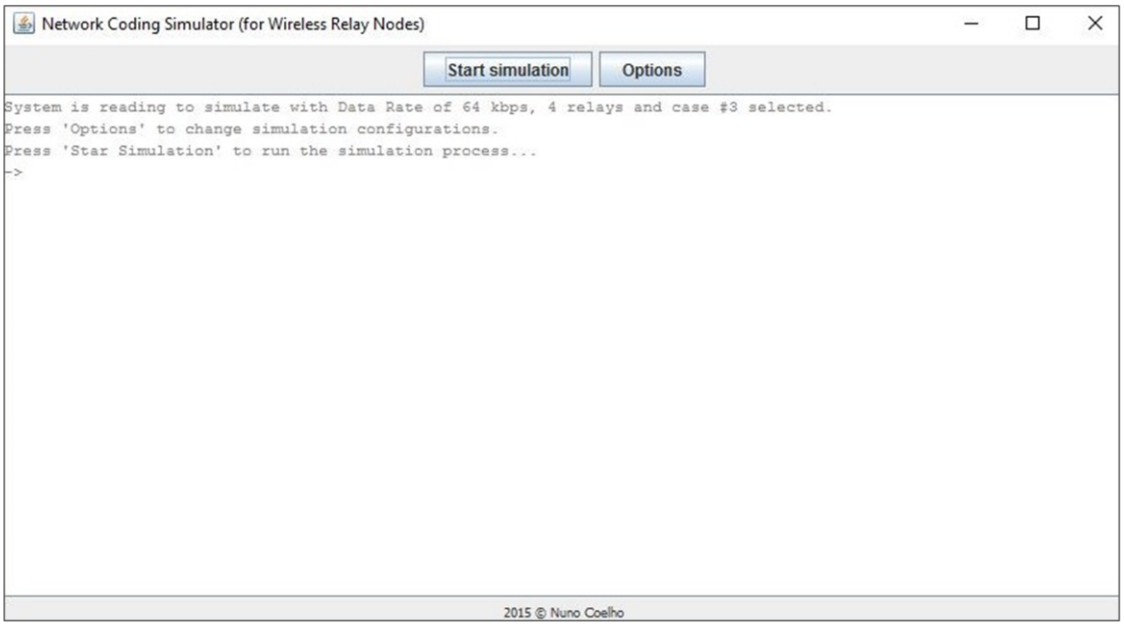
$$\mathbf{G} = \begin{bmatrix} P1 \\ P1 \oplus P2 \\ P2 \oplus P3 \\ P3 \end{bmatrix}$$

If an intermediate node fails, the destination will put in the matrix a “redundant” packet, only to complete the matrix. We assume that P_1 and $P_1 \oplus P_2$ are neighbours because their matrix line distance is 1. For example, to decode $P_1 \oplus P_2$, the algorithm works by finding the active neighbour with a lower coding level. In this case, it ends up finding P_1. We assume that P_1 and P_3 coding level is 1. $P_1 \oplus P_2$ and $P_2 \oplus P_3$ coding level is 2. The coding level of each packet is directly related with its intermediate node. The destination knows the coding level of each packet through the encoding information contained inside the packet itself. This encoding information is added by the intermediate node and it is simply the **H** matrix line that represents the intermediate node itself. An active neighbour is a “packet” whose intermediate node is active. We know the intermediate node state (active or not) from the **H** matrix. An intermediate is active when its respective line in the **H** matrix has at least one ‘1’. When no more lower coding level neighbours exist, and there are still packets to decode, the algorithm works differently: it starts to decode the already decoded packets with neighbours with equal coding level.

The destination stops decoding when it decodes a number of packets equal to the number of sources given by matrix **H**. When the emulation time ends, the results are presented to the user.

The emulator shows the number of packets sent by the sources, the intermediate nodes state, the network details and other information parameters.

These earlier versions had a quite rude graphical user interface (GUI), as shown in Figure 3.7, because at that point the interface of the emulator was not important.

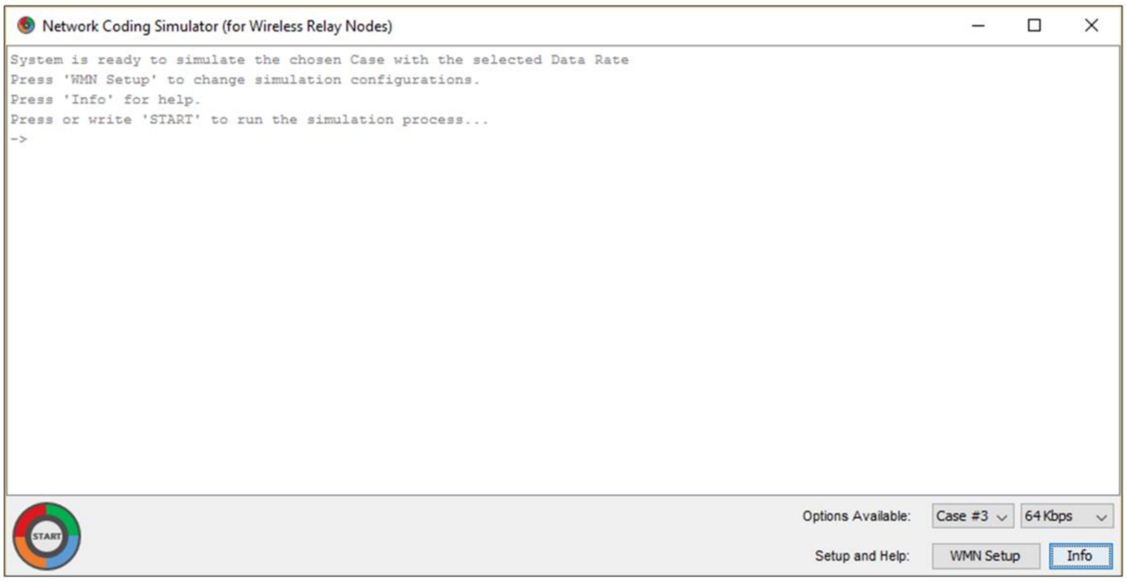


```
System is reading to simulate with Data Rate of 64 kbps, 4 relays and case #3 selected.
Press 'Options' to change simulation configurations.
Press 'Star Simulation' to run the simulation process...
->
```

Figure 3.7 - Emulator GUI v1.2.

3.5 - Source-intermediate Matrix Approach

In some versions of the emulator the main changes were made to the emulator's interface, as shown in Figure 3.8, and to the decoding algorithm. The available options have also been changed, allowing the user to change some parameters of the emulation, such as the source's data rate.



```
System is ready to simulate the chosen Case with the selected Data Rate
Press 'WMN Setup' to change simulation configurations.
Press 'Info' for help.
Press or write 'START' to run the simulation process...
->
```

Figure 3.8 - Emulator GUI v2.0.

The second decoding algorithm is much simpler because it is simply based on the fact that one wants to solve a linear system. The main reason why this algorithm had not been used in the earlier versions is that a Java library had not been found to compute matrix operations in *GF*(2). This obstacle was overcome with the N-ML approach. In these versions we found a Java library [36] that allows do matrix operations in the binary Galois Field. The reason why one needed to perform matrix operations in *GF*(2) is obviously because one was working with the "0" and "1" symbols. We call this second decoding methods the source-intermediate matrix (S-IM) approach, which is a similar method to the one used in the coding process. We know that for coding, the packets sent by the intermediate nodes to the destination is defined as **G**=**HX**. So, in order to decode the received packets **X**, the destination needs to solve this linear system, and thus retrieve the information. The S-IM approach performs the following operation:

$$\mathbf{X} = \mathbf{H}^{-1}\,\mathbf{G} \tag{3.2}$$

3.6 - QoS parameters and Information Presented

The analysis of the emulation is very important to understand the behaviour of the network when we implement some ideas, in order to understand their advantages and their disadvantages. In order to analyse the emulation, the user needs to see the emulation results. The emulation QoS parameters analysed and also presented in the emulation results are the packet loss ratio (PLR), latency and jitter. The PLR only assumes values between 0 and 1. This parameter is calculated in `calcPLR()` emulator function in *Network.Java*. These parameters are calculated as follows:

$$\text{PLR} = 1 - \frac{\text{Total number of packets received by R}}{\text{Total number of packets sended by all sources}}\ , \tag{3.3}$$

$$\begin{aligned}\text{Latency} = {} & (\text{propagation delay} \times \text{jumps}) + (\text{transmission delay } \times \text{jumps}) \\ & + (\text{switching delay} \times \text{jumps})\ ,\end{aligned} \tag{3.4}$$

$$\text{Propagation delay} = \frac{d}{V}\ , \tag{3.5}$$

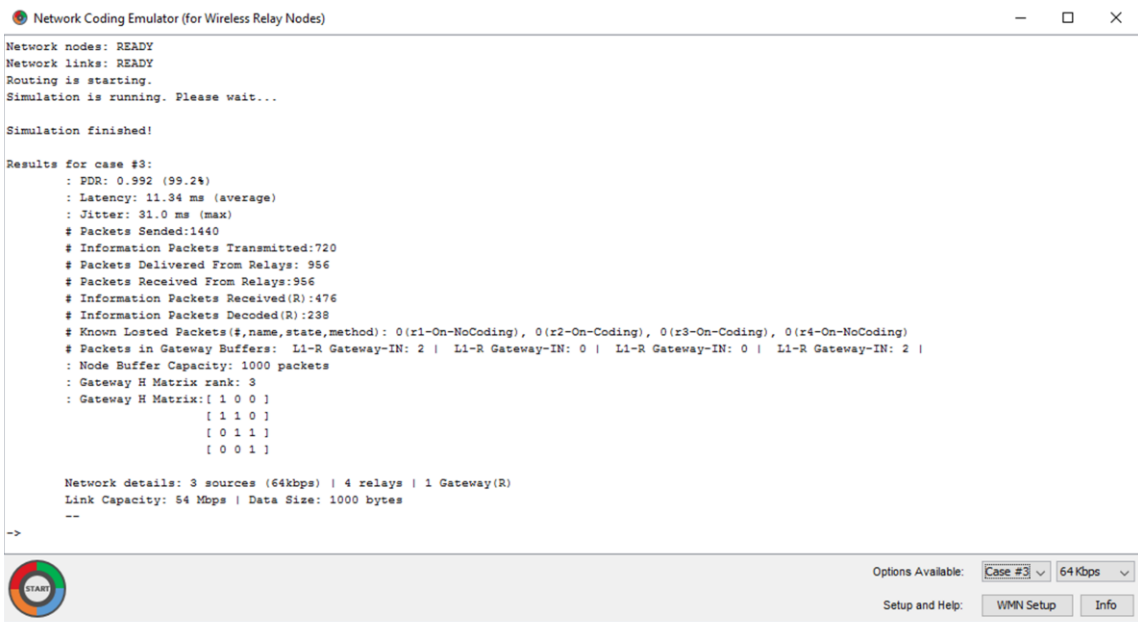
$$\text{Transmission delay} = \frac{L}{R}\ . \tag{3.6}$$

The latency is the sum of the propagation delay, the transmission delay, and the switching delay. The switching delay is given by the emulator and represents the total computing time, i.e., the total time that the node takes to compute the packet, since it arrives to the node until it is released at the output. Both the propagation delay and the transmission delay are theoretical variables. We consider *d* as the distance between the network nodes, *V* is the propagation velocity, *L* is the packet size and *R* is the link capacity. We consider 30 meters as the distance between nodes and 3×10^8 *m/s* as the propagation velocity.

We assume jitter as the maximum difference between the packet arrival time instant and the packet departure time instant. This parameter is calculated in `calcJitter()` in *Network.Java*. Let *R* be the destination switching time instant and *S* be the source switching time instant for both packets *i* and *j*. Jitter is calculated as follows:

$$\begin{aligned} D_{i,j} &= T_j - T_i \ , \\ T_i &= R_i - S_i \ , \\ \text{Jitter} &= \max_{i,j} |D_{i,j}| \ . \end{aligned} \tag{3.7}$$

Some other important information is also presented to the user, namely: the number of packets sent by the source nodes, the number of packets decoded by the destinations, the packet structure, and the number of generations of each source (see Figure 3.9).



Network Coding Emulator (for Wireless Relay Nodes)

```
Network nodes: READY
Network links: READY
Routing is starting.
Simulation is running. Please wait...

Simulation finished!

Results for case #3:
        : PDR: 0.992 (99.2%)
        : Latency: 11.34 ms (average)
        : Jitter: 31.0 ms (max)
        # Packets Sended:1440
        # Information Packets Transmitted:720
        # Packets Delivered From Relays: 956
        # Packets Received From Relays:956
        # Information Packets Received(R):476
        # Information Packets Decoded(R):238
        # Known Losted Packets(#,name,state,method): 0(r1-On-NoCoding), 0(r2-On-Coding), 0(r3-On-Coding), 0(r4-On-NoCoding)
        # Packets in Gateway Buffers:  L1-R Gateway-IN: 2 |  L1-R Gateway-IN: 0 |  L1-R Gateway-IN: 0 |  L1-R Gateway-IN: 2 |
        : Node Buffer Capacity: 1000 packets
        : Gateway H Matrix rank: 3
        : Gateway H Matrix:[ 1 0 0 ]
                           [ 1 1 0 ]
                           [ 0 1 1 ]
                           [ 0 0 1 ]

        Network details: 3 sources (64kbps) | 4 relays | 1 Gateway(R)
        Link Capacity: 54 Mbps | Data Size: 1000 bytes
        --
->
```

Options Available: Case #3 | 64 Kbps

Setup and Help: WMN Setup | Info

Figure 3.9 - Emulator v2.3 console results.

In this version, a sample of the routing of packets between a source node and a destination node is also presented, to prove the correct decoding of packets. Besides this information, the user may see the network details, e.g., the links' capacity and the data rate of the source nodes.

3.7 - Emulator validation

In order to validate the emulator, the emulator development was based in the network models and parameters of the authors in [8]. We have replicated the study cases (cases 3-6) of these authors in order to calibrate and, therefore, improve the emulator. The emulator calibration and validation is a very important step towards the objectives of this work. Although calibration is a continuous process to improve the emulator, the validation is not. It is very important to guarantee that the emulator presents reliable results. We have analysed three QoS parameters, packet delivery ratio (PDR), latency and jitter. PDR is the ratio of total number of packets received at the destination node over the total number of packets sent by all sources. Our results are very good and very similar to the authors in [8]. Notice that for each case we are presenting the average results of 30 emulations.

Our PDR values were always between 87% and 96%, as shown in Figure 3.10 while the author's PDR results were between 99% and 66%, as shown in Figure 3.11.

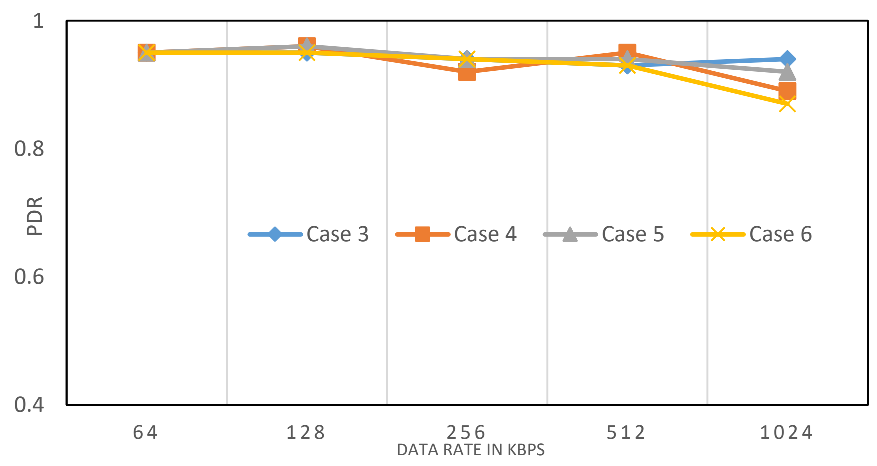


Figure 3.10 - Results of PDR for the first emulations

For cases 3-5 at most one failure occurs. In case 6 two failures occur. Analysing our results, it's clear that the results are quite similar. In all cases, one obtains a better PDR (≈96%) for lower data rates and it reduces for higher data rates (≈87%). The higher PDR value was 96%, obtained in cases 4 and 5. The lower PDR values was 87%, obtained in case 6. In case 4 and case 5, even though one intermediate node fails, the destination still retrieves all packets. In case 6 the same happens; even though two relay node fail, the destination node still retrieves all packets without any information loss. This PDR results are quite similar to the ones in [2]. In all cases, the destination still retrieves all packets but the PDR is always less than 100% due to relay nodes failures.

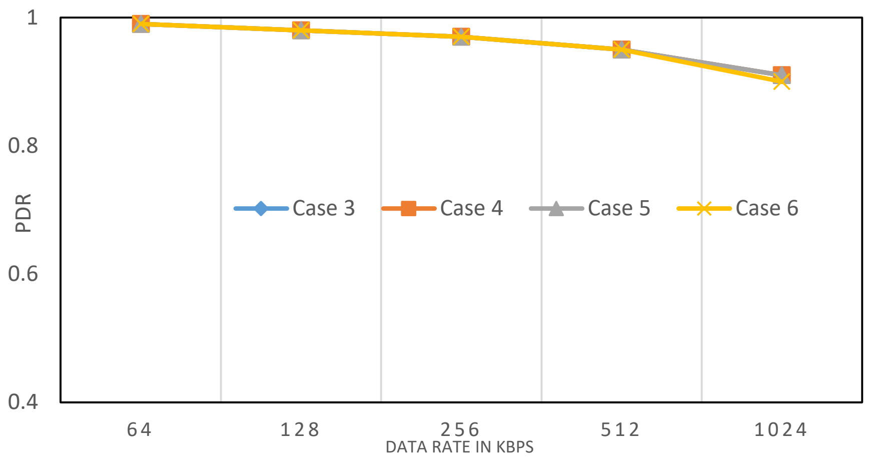


Figure 3.11 - PDR results of *[8]* for cases 3-6.

Our latency results were between 7 *ms* and 14 *ms* while the author's latency results were between 50 *ms* and 10 *ms*, as shown in Figure 3.12 and Figure 3.13 respectively.

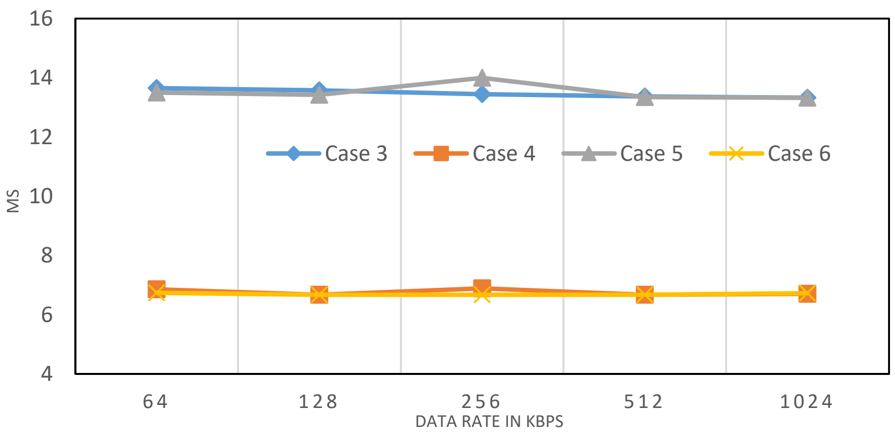


Figure 3.12 - Results of Latency for the first emulations.

Independently of the data rate, the latency results remain fairly constant. In cases 4 and 6 the latency values (≈7 *ms*) is lower than in cases 3 and 5 (≈14 *ms*). The latency results from [8] are different; there, the latency values remains constant as the data rate increases. Moreover, in [8] the latency decreases from ≈50 *ms* to ≈10 *ms* as the data rate increases. In our results the latency decreases but with less intensity.

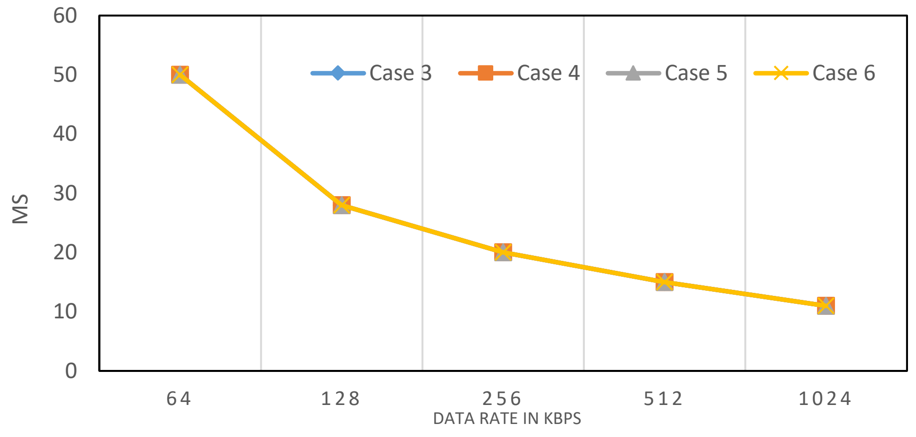


Figure 3.13 - Latency results of [2] for cases 3-6.

The jitter results were very different. While the results in the emulator are around 20 *ms* (see Figure 3.14), in [4] jitter is shown to be about 0.5 *ms*, as shown in Figure 3.15.

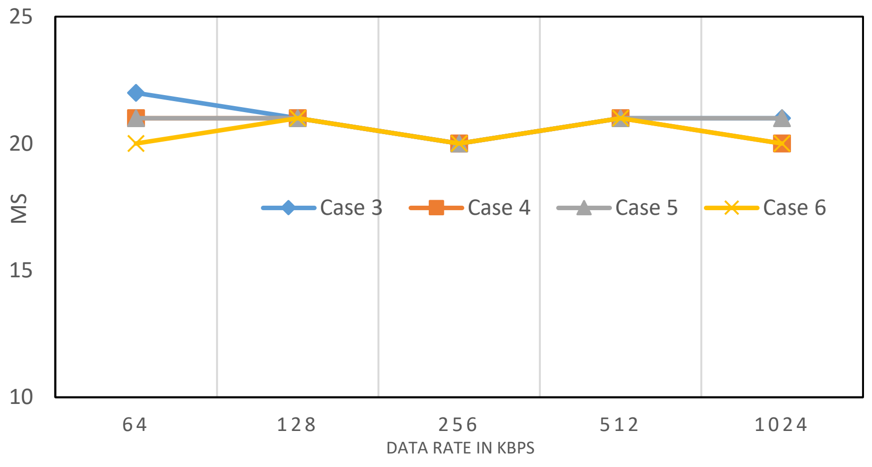


Figure 3.14 - Results of jitter for the first emulations.

Observing the results, one can see that the jitter values remain similar in all cases (≈20 *ms*). This results are rather different from the results in [2] (≈0.5 *ms*). However, the definition of jitter used in [2] is not available in that paper and therefore one cannot jump to any conclusion regarding this difference.

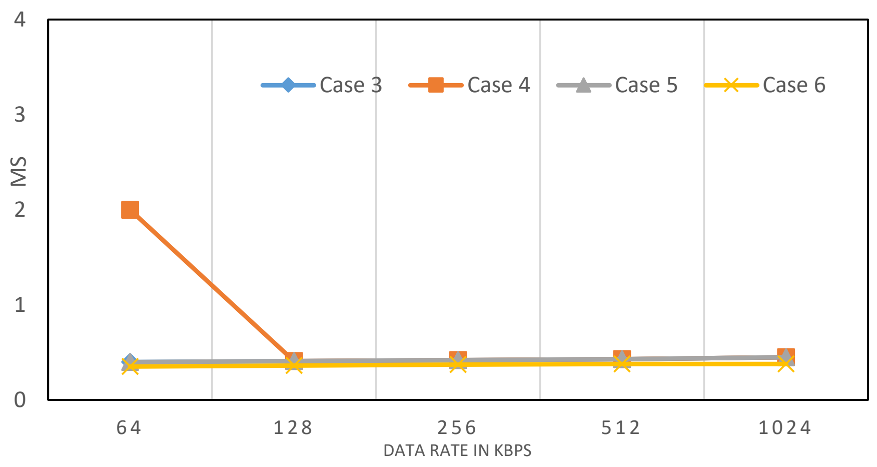


Figure 3.15 - Jitter results of [2] for cases 3-6.

Moreover, the results obtained with the emulator are also not similar to the results in [8]. These later comparison was important to assess some emulator’s strengths and weaknesses. In fact, it was an important step towards improving the techniques used, and try new methods.

One of the weaknesses identify was the inflexibility of the emulator that did not allow the user to configure the network or the coding methods. It is very important to ensure some configuration flexibility in order to allow the user to test several scenarios. The biggest advantage that was found was the adaptation capability of the emulator as changing any component of the emulation is very simple. Another advantage was to observe PDR results which allowed not only to validate the emulator but also to ensure that the routing of data was being correctly done. On the downside, the jitter results and the latency results did not match the expected ones. It should be noted that jitter results are very hard to compare because there is not one single formula to define jitter. The latency results depend on the nodes capacity, buffer capacity and links capacity. The results obtained with the emulator are not necessarily worse than the results in [8]. In fact, given that the exact definitions are not available in [4], the results are very hard do compare.

Chapter 4

Flexible Network Coding

The present chapter explains the ideas developed during the work. It also presents the emulator modes, the coding and decoding technique, and the numerical results obtained.

4.1 - Requirements for a Flexible Emulation

Once the emulator validation is completed it was decided to implement new ideas in order to eliminate the detected weaknesses and to assess new methods. To that end, one must potentiate the emulator's capabilities, allowing the user to change several scenarios and making the emulator much more flexible.

As it is well known, there exists a sustained growth of data traffic that are used by applications. These applications requires flexible and efficient networks (e.g., cloud computing [37]), which is one of the reasons why it was important to introduce much flexibility in the emulator. Implementing network coding in the emulator is simple because it does not impose many extra requirements, once one has a flexible emulator. The emulator components must be able to handle several types of scenarios specified by the user or by the emulator itself. One can define here two types of flexibility:

- Network topology – the emulator is capable of emulating several types of networks given that the network is correctly configured;
- Node coding method – the emulator is capable of emulating a network with different coding methods in the intermediate nodes.

The users should only need to define the network topology, and then the linear model defined in the precious chapter suffices to emulate the routing of the user-defined network while using NC. Since we are working at an abstract level with layer independent and an abstract type of network, there is no need to define the type of nodes or configure anything beside the physical connections between nodes.

The user has the capability of defining the physical links between nodes by manually configuring the network nodes in the emulator setup or by using the emulator console commands. The user can choose the number of nodes and *how* they are connected. Alternatively, a pre-set configuration can be chosen when a quick setup of the emulator is needed. The network is distributed and operates according to the physical connections, whether it is a manual configuration or a pre-set configuration.

The emulator's console can only be initiated if the network is correctly configured, this is, if the configuration has no errors. An example of an error is the case when the network destination is not connected to any other network node. For that reason, *probing packets* are used, as it will be further explain in the following.

4.2 - The Concept of Generations

After improving the emulator several times, it was decided to change the way nodes encode and decode the packets. As common in software projects, the early versions of the emulator were pilot project in which some ideas were developed, having in mind the main objective of building a functional tool to assess LNC. Those early versions were mainly based on other works in the literature. The third version of the emulator became essentially the final one.

The first improvement was to change the main idea of combining packets from different sources at the intermediate nodes. It was decided to only combine packets that came from the same source node. The idea of combining information from different sources would lead to other problems. One of those problems changes the initial purpose of NC, as illustrated in Figure 4.1. There will be unnecessary information reaching the different destination nodes that the receiver will not use.

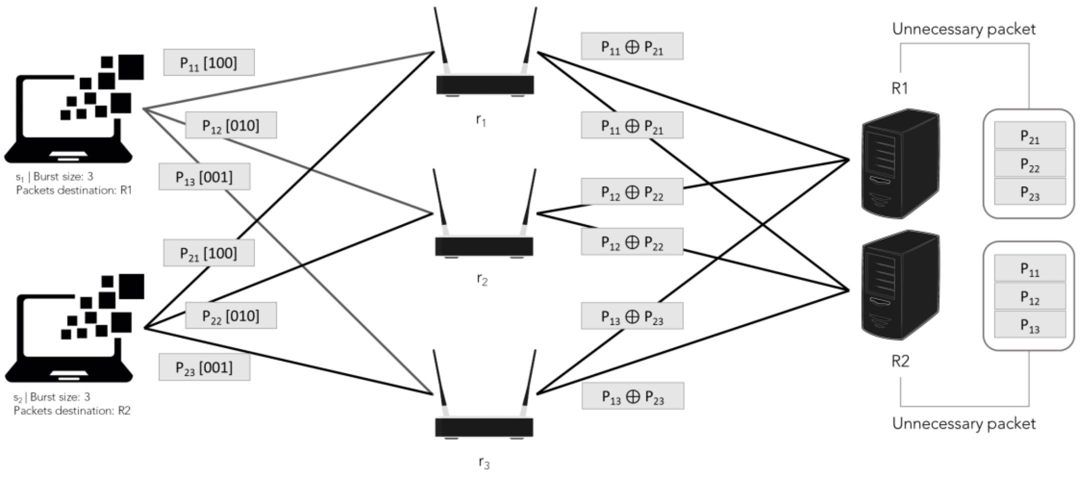


Figure 4.1 - Unnecessary information routing.

Another problem that arises when mixing packets from different sources is the problem of having different sizes of transfer vectors. As said before, the size of the transfer vector is equal to the burst size. If two sources have different burst sizes, we cannot encode or decode the packets together, as shown in Figure 4.2. If in this two examples we ignore the linear independence of the system, we see that combining packets from different sources causes some problems that must be avoided. Hence, one should only combine packets that belong to the same source node, ensuring a more efficient data routing, avoiding problems that go against the initial purpose of the network coding method. To achieve this, the packets cannot be randomly coded. Only packets with the same generation number are combined together.

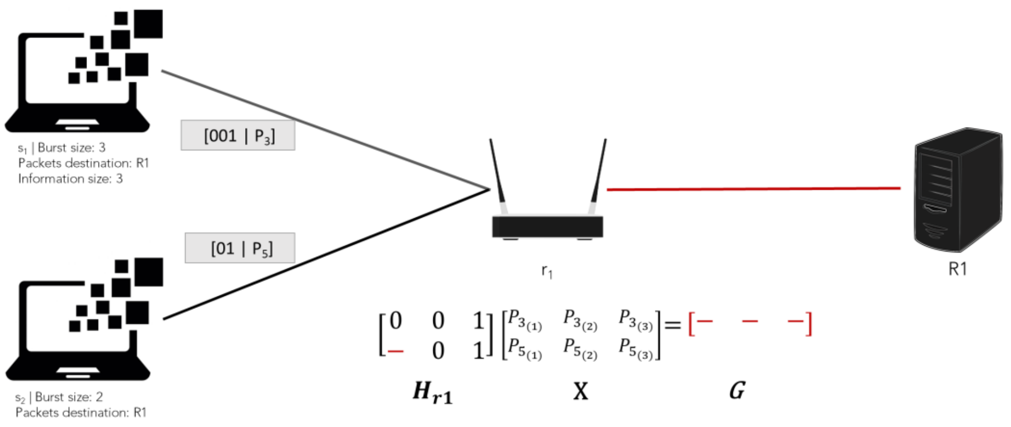
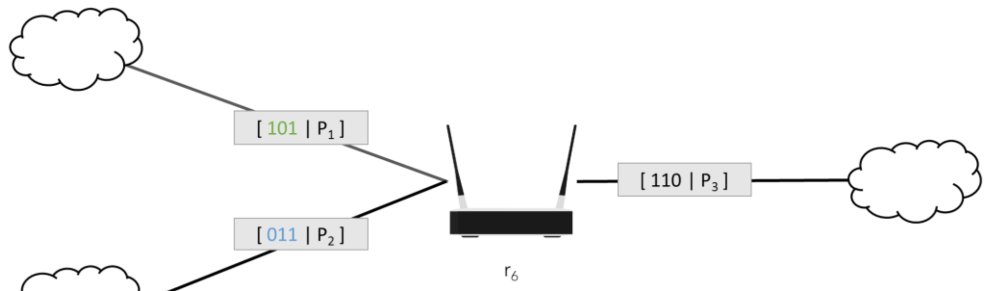

[001 | P3]
s1 | Burst size: 3
Packets destination: R1
Information size: 3
[01 | P5]
s2 | Burst size: 2
Packets destination: R1
r1
R1
$\begin{bmatrix} 0 & 0 & 1 \\ - & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{3(1)} & P_{3(2)} & P_{3(3)} \\ P_{5(1)} & P_{5(2)} & P_{5(3)} \end{bmatrix} = [- \quad - \quad -]$
H_{r1}
X
G

Figure 4.2 – An example of “bad coding”.

As defined in chapter 3, the generation number is directly related with the burst size. Let one consider that the intermediate node r_1 receives 5 packets: P_1, P_2, P_3, P_4 and P_5. Packets P_1, P_2, P_3 and P_4 belong to source s_1, which has a burst size of 3, and P_5 to source s_2, which has a burst size of 2. Notice that x may be 0 or 1. The first three packets belong to generation number 45 and the other two packets belong to generation number 47 and 4 respectively. Using the XOR scheme, r_1 will encode packets P_1, P_2 and P_3 $(P_1 \oplus P_2 \oplus P_3)$, because they all belong to the same generation, same source node and their count is equal to the burst size of their source node. In order for a destination node to correctly decode a generation of packets, it must receive m linearly independent transfer vectors from each generation [38], where m corresponds to the number of packets of a generation, i.e., the burst size.

Instead of having a unique transfer matrix (**H**), the intermediate nodes and the destination nodes now have a *dynamic* **H** matrix. This dynamic transfer matrix is similar to the previous one, used since the first version of the emulator over $\mathbb{F}_2$. The difference now is that **H** changes several times during the emulation, according to the packets to be coded. As we said before, each packet carries a transfer vector in their message body. This transfer vector will be used to build the **H** matrix of each intermediate node. **H** will be composed only by one line, resulting of the combination of the transfer vectors of the packets received, as shown in Figure 4.3, where one may see how the transfer matrix is composed for intermediate node r_6 after receiving packets r_1P_{56} and r_5P_{36}. Basically, the **H** is now a vector that combines the transfer vectors of the packets that will be coded. Multiplying **H** by **X**, will result in the coded packet that will be sent to the next network node. The transfer vector of the coded packet will be the transfer matrix of that node. This means that the result of coding the packets represented by the rows of a matrix **X** at an intermediate node will results in one coded packed. In the example in Figure 4.3, in order to perform the operation (multiplication) a padding line is added to the **X** matrix. This padding line has no influence on the coded packet.



$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} = \boldsymbol{H_{r6}}$$

$$\boldsymbol{G} = \boldsymbol{H_{r6}}\,\mathrm{X} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_{1_{(1)}} & P_{1_{(2)}} & P_{1_{(3)}} \\ P_{2_{(1)}} & P_{2_{(2)}} & P_{2_{(3)}} \\ 0 & 0 & 0 \end{bmatrix}$$

Figure 4.3 - Construction of H matrix in intermediate nodes.

Matrix **H** is quite simple to calculate, as it will be shown. Similarly to what was being implemented in the previous versions of the emulator, the source nodes send packets to the destination nodes. The only difference now is the inclusion of a generation number and a burst size. The concept of generations allow one to group the packets according to their generation number. The burst size will determine the number of packets of each generation. This is exemplified in Figure 4.4: if s_1 has a burst size of 2, the number of packets of one generation will be 2, thus, the first two packets of s_1 will belong to generation number 2.

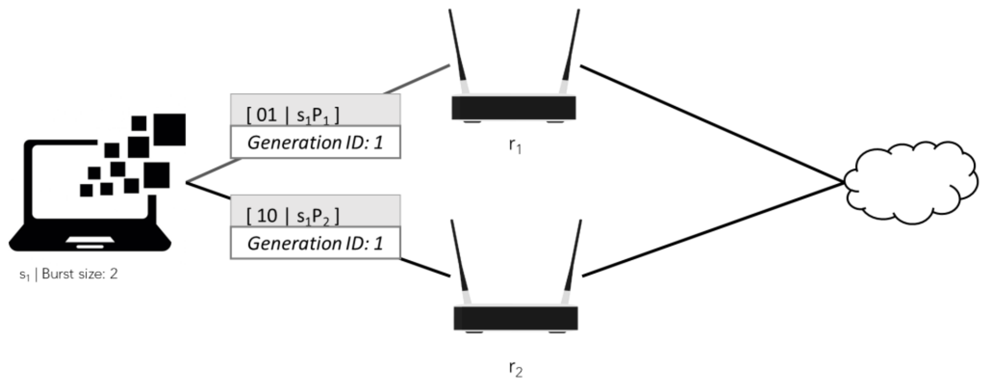


Figure 4.4 - Packets generation number.

4.3 - Buckets Computation

It is important to describe how the destination nodes operate when they receive a packet. As shown in the following pseudo-code, the destination only uses a received packet when the packet “adds value” to the existing stored information at that destination.

Algorithm 1: Algorithm that describes a destination operation when receiving a new packet.

```
input:  Packet : ∀p ∈ Packet
        (any network packet)

input:  Link : ∀l ∈ Link
        (input link associated to the node)

inBuffers : the node list of out buffers

out ← ∅
add packet p to buffer of link l // adds the packet to the buffer correspondent of l
foreach b of inBuffers do
   foreach p2 packet of b do
        if there is a bucket created for p2 source node, with p2 generation number and p2
        destination node then
             if b increases rank then
                add p2 to b
             end
        else
             create new bucket for p2
        end
   end
end
```

According to Algorithm 1, when a new packet is received, the destination places it in a buffer. Secondly, there is an iteration in each buffer in order to put the existing packets into buckets. Each source’s generation has a bucket, containing only packets of that source’s generation. When a packet is in a buffer the destination needs to put it into an existing bucket or in a new bucket. Assume that packet *p* arrives to destination *g*. Packet *p* belongs to the second generation of source *s*. When *g* receives the packet, it will check if there is a bucket created for the generation of that source. If there is not a bucket yet created, the destination will create a new bucket and put the packet in it. If there is a bucket created, *p* will be added to the existing bucket if it “adds value”. to the existing bucket, i.e., if the bucket rank increases or reaches the desired rank. A bucket desired rank is equal to the burst size, i.e., is the number of packets of the generation. The bucket rank is calculated with the **H** matrix, since the matrix is composed by the transfer vectors of the bucket’s packets.

4.4 - User-defined Networks and Emulator Modes

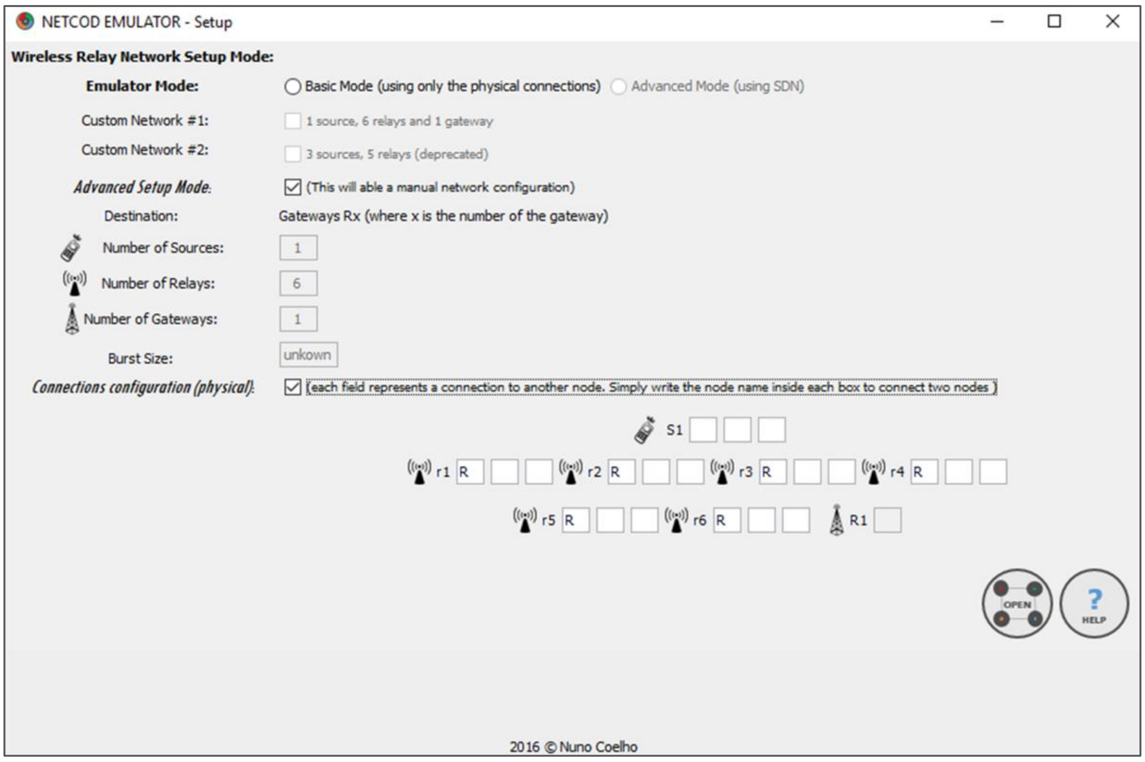
In the new versions of the emulator, we also generalized the network topology, allowing the user to choose the topology of the network. The user has the capability of defining the physical links between nodes by manually configuring the network nodes in the emulator setup, as showing Figure 4.5. The user can choose the number of nodes and define how they are connected or he or she can choose a pre-set configuration, if he or she prefers to quickly start an emulation process.

There are two emulator modes: the basic mode and advanced mode. The basic mode is the simplest one. The network is constructed and operates according to the physical links, whether is a manual configuration or a pre-set configuration. The network is distributed, so each node only knows its neighbours. In this mode the user may choose two approaches:

- The traditional approach – the network nodes are not coding or decoding, i.e., the intermediate nodes are simply forwarding the packets choosing the shortest route, the source is sending native packets and the destination nodes are simply receiving the packets;
- The network coding approach – the nodes are using LNC to code and decode the packets, i.e., the source nodes and the intermediate nodes are coding the packets and the destination nodes are decoding the packets.

In traditional packet-based networks, the intermediate nodes have either dynamic or static routing tables. The routing tables are built based on a metric or in a combination of metrics and this may generate one or more routes. In the basic mode with the traditional approach, the shortest route is chosen by an algorithm that checks the route to the destination node with fewer jumps. Therefore, assume that the number of jumps is our metric.

In the advanced mode of the emulator, the network is centralized, i.e., there is an entity that controls the network. In the advanced mode the network is genie-based, i.e., there is *a genie* (or *genius*) that controls all nodes. In this mode, the approach is always the centralized network coding approach. In both modes, the emulator console can only be initiated if the network is correctly configured, i.e., if the configuration has no errors (for example, the network destination is not connected to another network node.


NETCOD EMULATOR - Setup

Wireless Relay Network Setup Mode:

Emulator Mode: ○ Basic Mode (using only the physical connections) ○ Advanced Mode (using SDN)

Custom Network #1: ☐ 1 source, 6 relays and 1 gateway

Custom Network #2: ☐ 3 sources, 5 relays (deprecated)

Advanced Setup Mode: ☑ (This will able a manual network configuration)

Destination: Gateways Rx (where x is the number of the gateway)

Number of Sources: 1

Number of Relays: 6

Number of Gateways: 1

Burst Size: unkown

Connections configuration (physical): ☑ (each field represents a connection to another node. Simply write the node name inside each box to connect two nodes)

S1

r1 R

r2 R

r3 R

r4 R

r5 R

r6 R

R1

2016 © Nuno Coelho

Figure 4.5 - Emulator setup.

In the setup the user may choose which mode he or she wants to start the emulator. The emulator console only starts if there is no problems with the physical connections. By choosing the basic mode, the user is only relying on the physical connections, i.e., if the network does not provide at the destination a full-rank system of linearly independent equations, the emulation will not work. In the advanced mode, the network genius controls the network connections. In this mode, the network genius will configure the network physical connections in order to achieve the best network configuration possible. We defined the best network configuration based on three parameters:

- **Rank metric**: the system needs to be linear independent, i.e., each destination node must have an **H** matrix that is linear independent. A linear independent system guarantees that every message is correctly decoded;

- **Network cost**: the network cost is the number of links connected, i.e., the number of links that are being used by nodes. Each link connects two nodes.

Ensuring a linearly independent system of equations is the hardest requirement to be met, once it depends not only on the connection matrices of the intermediate nodes but also on the other parameters. The network cost is quite easy to compute, however, sometimes it is not easy to make the cost smaller because in doing so other parameters may change and may turn the routing of data impossible.

Based in these two parameters, the best network configuration is the one that reaches the lowest network cost always guaranteeing a linearly independent system and a reliable network. A reliable network is hard to obtain once it depends on these three parameters. Notice that by decreasing the network cost, the probability of ensuring a reliable network becomes smaller, as protection to link failures becomes harder to assure. Note the problems in the next example shown in Figure 4.6, when trying to decrease the network cost.

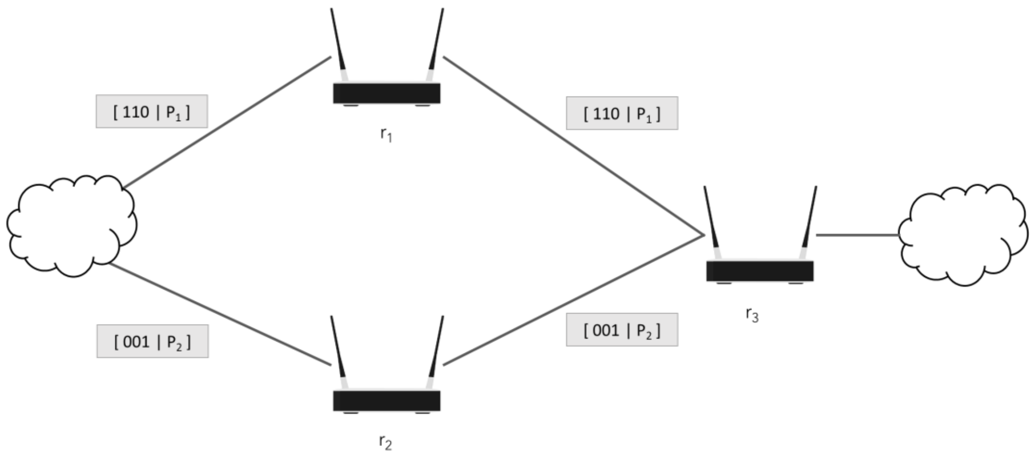


Figure 4.6 - Low network cost problems.

Decreasing the network cost may lead us to a rank deficiency. In the example above the packets P_1 and P_2 are being transmitted to their destination. P_1 is a coded packet with two native packets inside and P_2 is a native packet, the third of a generation of size three. Decreasing the network cost in this network by cutting any of the links will lead the network to be rank deficient at the destination nodes, i.e., to a linear dependent system, because when at least one link is cut P_1 or P_2 will not arrive at their destination. It is very important to decrease the network's cost because it will reduce the burden in the network use, but linear independence cannot be compromised.

4.5 - Rank Metric for a Viable Network

When a user sets up a determined configuration, by manually configuring the network physical communications, we need to guarantee that the network is viable, this is, that the following criteria are met:

- **Connection** – each destination node must be connected to at least one source node;
- **Linear independent system** – each destination node must have an **H** matrix that is linearly independent.

To ensure the first criteria, connection, a recursive algorithm was developed, as presented below. This algorithm ensures that every destination node is connected to at least one source node, by recursively checking if a network node is connected to source node.

Algorithm 2: Recursive algorithm for checking node connection extreme-to-extreme.

```
method checkSourcesConnectivity()
   inBuffers : the node list of in buffers
   out ← ∅
   foreach b of inBuffers do
      get node n from the link of b
      find recursively a source node originated in b, searching on n(findSource method)
   end

method findSource (Node n, Buffer b)
   sourcesFound: an temporary list of source nodes
                 found in the node
   input:  Node : ∀n ∈  Node
           (any existing type of network node)
   input:  Buffer : ∀b ∈  Buffer
           (any buffer)
   out ← ∅
   foreach b of inBuffers do
      if n is an intermediate node then
         foreach l of node n links do
            get node n2 from link of b
            if n2 is not n then
               if n2 is a source node then
                  adds n2 to the sources found
               else
                  find recursively a source node originated in b, searching on n2
               end
            end
         end
      else if n is a source node then
         adds n2 to the sources found
      end
      end
   end
```

Algorithm 2 is called in Network.Java by `checkNodesConnectivityToDestination()` method when the user starts the emulation. This method is called for all destination nodes of the network. If a destination has zero connections to a source node, the user is warned but he can still proceed with the emulation. If all destination nodes have zero connections to a source node, the user will also be notified that the emulation cannot proceed without reconfiguring the network. This method checks the first criteria for guaranteeing that the network is viable. The method checks every entry link on each node in order to find a source node. It starts on a destination node than it checks its entries to find the next connected node. If the next node is an intermediate node, the algorithm will search source nodes through that node’s entry links. If the next node is a source node the algorithm starts finding in the next entry link of the destination until there are no more entry links to check. So, the algorithm will be starting in a destination node and searching through the connected links for source nodes, ensuring that the destination is connected to at least one source node.

Algorithm 3: Algorithm for checking linear system independence.

```
input:  Bucket : ∀b ∈  Bucket
        (any bucket)
out ← ∅
get matrix H from the transfer vectors inside the packets of b
define integer desiredRank as the burst size of b
if rank of H is less then the desiredRank then
   notify user that the network is rank deficient
   stop emulation
else
   enable the emulation
end
```

Algorithm 3 will ensures the second viability criteria. This algorithm ensures that every destination node has a linear independent system when the physical network configuration is setup in the basic mode. In the beginning of the emulation in the basic mode a probing generation by every source to the destination. Consider a probing generation as being a group of probing packets. The objective of these probing packets is to check if the network is viable in terms of correctly decoding the message sent by the source node. Therefore, when the destination receives all the probing packets it will check if the rank of the linear system, **H** matrix, is the desired rank. The desired rank of the **H** matrix is equal to the burst size (i.e., the size of the generation), information available in each packet header, as presented in Figure 3.6. Notice that if the rank of **H** of a certain destination node it is not the desired one, the devised software will not emulate any network because linear independence was not guaranteed.

In the advanced mode there is no need to send probing packets because the network genius knows every connection matrices of each node. Knowing all connection matrices from all intermediate nodes and the source's **H** matrix, the genius is capable of computing all matrices in order to get the **H** matrix of the destination nodes. Given that in basic mode the network is distributed and there is no genius that has full knowledge of the network, the source nodes need to send probing packets to their destinations in order to check linear independence. In the advanced mode, the network control is centralized and there is a genius that knows everything about the nodes, so instead of routing probing packets, it simply computes the coding matrix of each source with the connections matrices of the intermediate nodes and gets the **H** matrix of the destination nodes. In the advanced mode, the process of checking linear independence is done quicker than in the basic mode, as there is no need for emulating the propagation of probing packets. Beside this advantage, in the advanced mode, the genius does more than checking linear independence, as it will be explained in section 4.6 - .

4.6 - Genius

As we explained, in the advanced mode there is an entity called genius that has full knowledge of the network. Even if the user wants to change a connection matrix or test a network without changing any connections, the genius will work autonomously, i.e., will make all the changes in order to improve the network performance. So, instead of only checking if the system is linearly independent as in the basic mode, the genius will try to improve the network by changing the connections matrices, as one may see in the example in Section 4.7 - . Even if a network has a linear dependent system, the genius may be able to find a new configuration that allows the user to perform the emulation.

When a user sets up a network and the network nodes are correctly connected network coding may be not capable of implement. Although it can be correctly connected, the network may have decoding problems that occur when the **H** matrices at the destination nodes have the linearly dependent rows. It is not very easy to manually configure a network that has a linear independent systems so the emulator must be able to change the physical connections set up by the user in order to be capable of emulate the network.

In a small network testing all combinations, a brute force test, may be the right step to find the best configuration but in a large network that may not be possible because it will take too much time. In a large network we have to adopt other combination testing strategies. For larger networks we may choose between non-deterministic, deterministic or compound strategies. Non-deterministic strategies include heuristic strategies such as simulated annealing and random strategies. Deterministic strategies include instant strategies, e.g., Covering Arrays [39]. The third choice is based on combinations of non-deterministic and deterministic strategies, e.g., random with Covering Arrays. As we are making a flexible emulator we need to prepare the genius for small and large networks because the user may define a network with 5, 20 or more nodes. We have to test some combinations based on a recursive algorithm that is divided in several steps, as presented next.

Algorithm 4 changes and test several combinations for the network physical connections, building a logical network above the physical network. Firstly, the algorithm checks if the primary configuration is viable and then it will try to improve it. Even if the primary configuration is not reliable, the emulator is capable of finding a new configuration that may be reliable. To improve the network, the genius will perform 5 steps. The first step is to change the connection matrix of the nodes that are sending empty transfer vectors as output. Assume an empty vector as a binary vector only containing '0' value in all positions. Then, the genius will disconnect the links of the same nodes that are sending empty transfer vector as outputs. The third step will disconnect links randomly. The fourth step will test random combinations by changing the connection matrices of random selected connection matrices. The final step will be testing the configuration in which all connections matrices have only value ‘1’ in all positions.

Algorithm 4: Recursive algorithm to improve the network connections.

```
minimum cost : integer value correspondent to the minimum cost of the network which is
               the maximum cost at the beginning of the algorithm

maximum cost : integer value correspondent to the maximum rank found which is zero at
               the beginning of the algorithm

step         : integer value that initializes at zero


foreach source node then
   sH is the source node coding matrix
   foreach destination node then
      foreach intermediate node directly connected to a source node then
         creates a connection structure with the output (transfer vector generated) from
         the source node connected to each IN buffer of the actual intermediate node
      end
      foreach intermediate node connected to another intermediate node then
         creates a connection structure with the output (transfer vector generated) from
         the intermediate node connected to each IN buffer of the actual intermediate
         node
      end
      define H, the transfer matrix of the actual destination node based on the transfer
      vector inside the auxiliary connection structure created
      define integer desiredRank as the burst size of the actual source node
      if rank of H is higher then the desiredRank then
         if the actual configuration as better rating then the last saved then
            define lastConfig as the actual configuration
            save the network configuration
         end
      end
   end
end
if step is less than 5 then // five is the number of steps
   increases step variable
   tries another configuration by calling method improveConnections() and calls this
   method recursively until it finish testing, this is when step equals 5
else
   starts the emulation with the best configuration
   updates the best configuration
end
```

```
method improveConnections(){
   if step equals one then
      change the connections matrices that are sending outputs with empty transfer
      vectors, i.e., with vector only containing ‘0’ value
   end

   if step equals two then
      disconnect link connected to the intermediate nodes which has connections matrices
      that are sending outputs with empty transfer vectors
   end

   if step equals three then
      disconnect links randomly
   end

   if step equals four then
      try random combination of connection matrices
   end

   if step equals five then
      try putting all connections matrices with value ‘1’ in all positions, i.e.,
      sending to all outputs the maximum number of coded packets
   end

}
```

If the user selects the advanced mode of the emulator, then the emulation starts the genius will execute this algorithm. What happens is that every time the user starts the emulation for the same network, the genius will execute this algorithm, so the network configuration may change which allows the user to test the network with different configurations until the genius find the best configuration possible.

For the genius to accept a new configuration it must have a better rating than the last saved configuration. The configuration rating is based on the three parameters explain in 4.4 - , i.e., rank metric and network cost. The configuration rating defined as CRa, which is the acceptance criteria is calculated as follows:

$$CRa = 70\% \times \frac{Actual\ Rank}{Max\ Rank} + 30\% \times \frac{Max\ Network\ cost}{Actual\ Network\ cost} \tag{4.1}$$

The rank criteria is the most important because it defines if the network is capable of correctly decoding the coded packets with network coding, so it has a weight of 70%. It is important to decrease the network cost in order to decrease the network use rate but it is not the most important thing when comparing with the other parameter. So this algorithm is based on the weight of the two most important parameters that ensure a reliable network using network coding. Typically, for an 8 to 16 nodes this algorithm is executed in less than 5 seconds which is not too long.

The genius saves the best configuration of the current network in a temporary structure that is represented by a class with singleton pattern. A singleton class ensures that only one instance of a class is created with a unique point of access to the object itself, i.e., guarantees that only one structure is created for the many emulations of the current network. Also we choose to apply the singleton pattern to the genius class, i.e., per each instance of the emulator there is only one genius.

Also, when using the network coding approach in the basic mode or the advanced mode we are avoiding the routing of native packets in the network. The number of native packets circulating in the network is very important. Avoiding native packets in the network will lead to a better secure network, due to secrecy and reliability questions already studied by other authors as we described in Chapter 2. So, we have changed the **H** matrix of the source nodes to code the packets directly on the source, avoiding the routing of native packets in the network, as we may see in the following section.

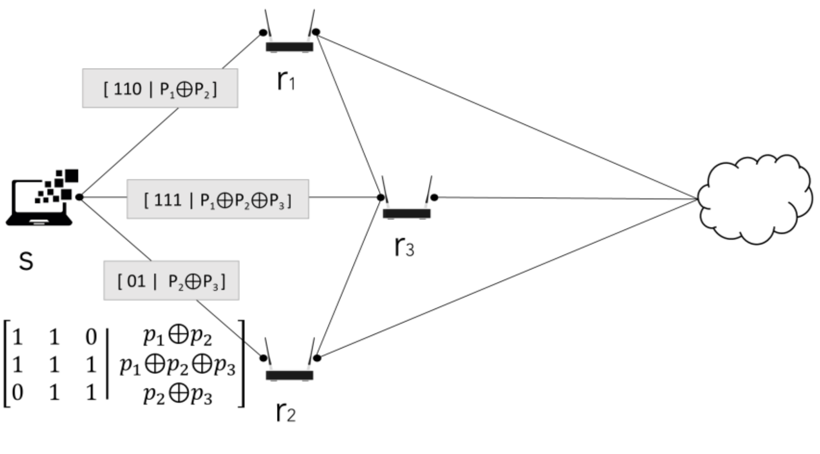
4.7 - Coding at the source

As we explained before, to encode the packets, the source uses the linear independent matrix **H**. **H** is a $m \times m$ matrix over $GF(2)$, where m is the burst size of the source, i.e., the number of packets of each generation. In the traditional approach **H** is an identity matrix which means the packets are not coded, i.e., the source node are sending native packets. In the advanced mode, the source are coding the packets before sending them to the network. For each emulation, an **H** matrix is generated for each source. **H** must be linear independent, i.e., the rank of H must be equal to the burst size of the source node. Also, each line of **H** must have a coding level higher than one. Assume coding level as the number of '1s' found inside vector or a line of a matrix.

For example, a source *s* with a burst size of three would have an $\mathbf{H}_s$ matrix like the following one, where p_1, p_2 and p_3 are native packets:

$$\mathbf{H}_s = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

If the source *s* wants to send packets p_1, p_2 and p_3 over $\mathbb{F}_2$, the initially transmit packets can be stacked and create $\mathbf{X}_s$. So, instead of sending the native packets, the source is sending a combination of them, avoiding the routing of native information in the network, as we may see in Figure 4.7.



$$\mathbf{X}_s = [\mathbf{H}_s|\mathbf{P}] = \left[\begin{array}{ccc|c} 1 & 1 & 0 & p_1 \oplus p_2 \\ 1 & 1 & 1 & p_1 \oplus p_2 \oplus p_3 \\ 0 & 1 & 1 & p_2 \oplus p_3 \end{array}\right]$$

Figure 4.7 – Coding at the source example.

4.8 - Coding and decoding

Developing a Software-Defined Network can be very complicated. Firstly we need to understand what is it and how it works, because SDN characteristics must not be in conflict with the emulator characteristics. SDN architecture is very dynamic and adaptable. Basically it enables the possibility of programming the network. We want a SDN that works like a real network. Must be adaptable, programmable, and dynamic with emulator identity.

We needed to implement the following SDN characteristics in the emulator:

- **Programmable** – the network control must be indirectly (must be automatically) programmable by an administrator that works like a “Genius”, knowing everything about the network and having the possibility of changing almost everything;
- **Adaptable and Agile** – if the network is programmable, it must be very easy to adapt. When a change is in the network configurations, the network must implement the changes very quickly;
- **SDN design pattern** – there may be a difference between the physical connections of the network and the logical connections of the network, where the first ones cannot change and the second ones are controlled by the network Genius;
- **SDN with emulator pattern** – the network programmable control cannot be in conflict with the emulator principles, this is, it must not be possible to programme an unrealistic configuration.

Until now each node output link associated to an OUT buffer was not configurable, this is, all the network nodes output links worked in the same way. That’s the basic mode of the emulator, each node sends the same information for its outputs, by combining the maximum number of packets it can.

The packets arriving at the *j*-th input of a node *N* result from the concatenation of a transfer vector with its corresponding coded packet and those incoming packets have the form $\left[\mathbf{h}_{N,j}^{(in)}\middle|\mathbf{x}_{N,j}^{(in)}\right]=\left[\mathbf{h}_{N,j}^{(in)}\middle|\mathbf{h}_{N,j}^{(in)}\boldsymbol{X}_{s}\right]$, where $\mathbf{X}_{s}$ is a matrix having the original packets from the source places in the rows of the matrix. The row vector $\mathbf{h}_{N,j}^{(in)}$ describes “the memory” of all the linear operations performed at each node past visited in the past that transformed those original source packets into the coded packet that is arriving at the *j*-th input of node *N*. In fact, this is a result of using systematic network codes [40], hence, the left part of the packet tracks its previous path and directly reveals what is being combined in the coded message part of the packet.

Similarly, at the *i*-th output of node *N* one finds packets of the form $\left[\mathbf{h}_{N,i}^{(out)}\middle|\mathbf{x}_{N,i}^{(out)}\right]=\left[\mathbf{h}_{N,i}^{(out)}\middle|\mathbf{h}_{N,i}^{(out)}\boldsymbol{X}_{s}\right]$. The stacking of all row vectors $\mathbf{h}_{N,j}^{(in)}$ (which are the transfer vectors) creates matrix $\mathbf{H}_{N}^{(in)}$, the stacking of the row vectors $\mathbf{x}_{N,j}^{(in)}$ (which are the coded packets arriving at node *N*) generate matrix $\mathbf{X}_{N}^{(in)}$. Similarly, at the output of node *N*, one can build the transfer matrix $\mathbf{H}_{N}^{(out)}$, as well as matrix $\mathbf{X}_{N}^{(out)}$ with the set of output packets stacked as the rows of this matrix after a new coding operation is performed at that node. In matrix form, the operations performed at each node *N* are described by

$$\left[\mathbf{H}_N^{(out)} \middle| \mathbf{X}_N^{(out)}\right] = \mathbf{C}\ \left[\mathbf{H}_N^{(in)} \middle| \mathbf{X}_N^{(in)}\right], \tag{4.2}$$

where **C** is the connection matrix of the node, which describes the coding (or combining) process. Note that at after the coding takes place at a node, one has the output transfer matrix $\mathbf{H}_N^{(out)} = \mathbf{C}\,\mathbf{H}_N^{(in)}$ and the new set of vectors $\mathbf{X}_N^{(out)} = \mathbf{C}\,\mathbf{X}_N^{(in)} = \mathbf{C}\,(\mathbf{H}_N^{(in)}\mathbf{X}_s)$. The columns of **C** are associated to the node's inputs and the rows are associated to the node's outputs.

Let us now consider a simple example where **C** describes a node with one output and three inputs and is $\mathbf{C} = [1\ \ 1\ \ 1]$ over $\mathbb{F}_2$. With some simplification of the notation, assume that a node *N* received from source s_i the packets $\mathbf{x}_1$ = [11100], $\mathbf{x}_2$ = [00111] and $\mathbf{x}_3$ = [10101] with transfer vectors $\mathbf{h}_{x1}$=[100], $\mathbf{h}_{x2}$=[010] and $\mathbf{h}_{x3}$=[001] respectively, all belonging to the same burst or to the same "generation". One obtains:

$$\mathbf{H}_N^{(out)} = [\ h_{x1}(0) \oplus h_{x2}(0) \oplus h_{x3}(0)\ ,\ h_{x1}(1) \oplus h_{x2}(1) \oplus h_{x3}(1)\ ,$$

$$h_{x1}(2) \oplus h_{x3}(2) \oplus h_{x3}(2)]$$

$$\mathbf{X}_N^{(out)} = \mathbf{C}\,\mathbf{X}_N^{(in)} = [1 \quad 1 \quad 1] \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$= [0 \quad 1 \quad 1 \quad 1 \quad 0]\ .$$

Consider the example in Figure 4.7. At the destination one will have to infer the original source packets **P** from the end-to-end relation between the original packets **P** and coded packets **Y** that arrived at the destination, described by the global transfer matrix **H**, i.e., one has to solve $\mathbf{Y} = \mathbf{HP}$. This is involves performing one matrix inversion GF(2) at the destination *d*:

$$\left(\mathbf{H}_d^{(in)}\right)^{-1} \left[\mathbf{H}_d^{(in)} \middle| \mathbf{X}_d^{(in)}\right]. \tag{4.3}$$

In the advanced mode we will have the presence of the genius that will be controlling the logic network, as explained before. To control the communications between nodes, the genius is able to configure the outputs of each node. In the basic mode the connection matrix of each node is always the same, as we may see in the example in Figure 4.8, but the user may change the matrices by using console commands.

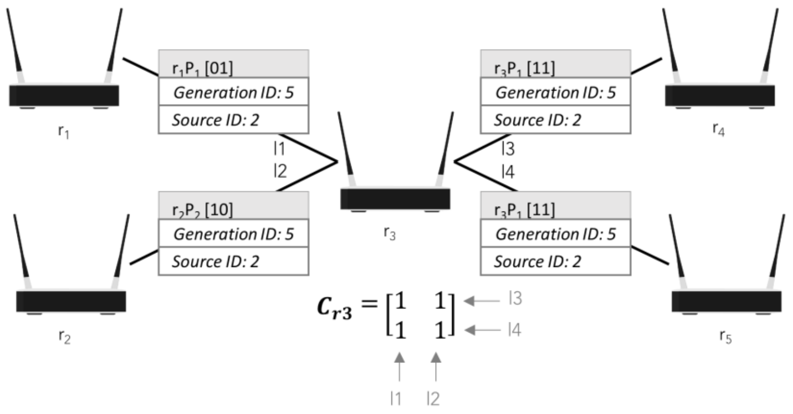


Figure 4.8 - Example of matrix **C** in Emulator basic mode.

In the example above r_3 will always combine packets from r_1 and r_2 and send the coded packet to r_4 and r_5. Notice that we are referring to inputs and outputs as the logical links connected to the node and not to the physical links. What is really important it's the link capacity. If we have two links, one with capacity *c* and another with capacity *2c*, we can say we have 3 logical links and when we look at our network we may see three links. To simplify, assume that logical link and physical link are the same because the connection matrix will not change if we assume one or another and the encoding model will be the same.

Initially each **C** matrix contains only value 1, which makes the output result of the advanced mode equal to the output resulting from the basic mode. Now, the genius may change the output of each output link, as we see in Figure 4.9.

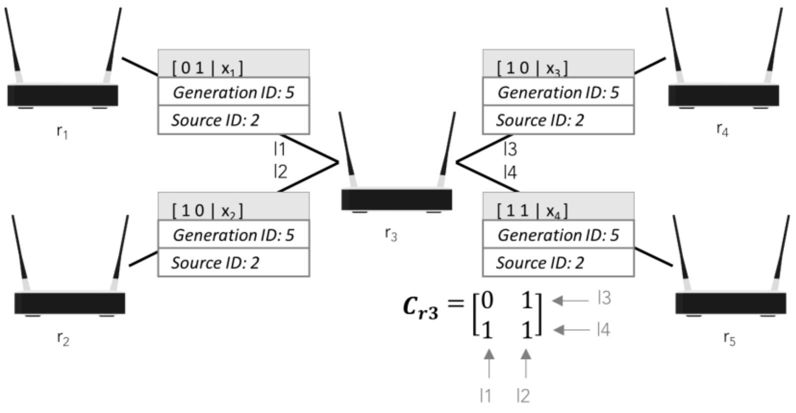


Figure 4.9 - Example of matrix C in Emulator advanced mode.

In the example above, there are two output nodes, r_4 and r_5, in the network and two intermediate nodes, r_1 and r_2, sending packets to intermediate node r_3. Genius can now (from version 2.6) change what is combined in each output. In this example, the intermediate node r_4 will receive packet x_3 (which in the example is in fact the same as x_2), and r_5 will receive $x_4=x_1\oplus x_2$. Notice that we are referring to inputs and outputs as the logical links connected to the node and not to the physical links.

This logical network configured by genius must have a better performance than the physical network since there was no point in wasting time configuring a network that is already setup. Performance in terms of robustness, e.g., packet loss ratio, latency and network cost. Genius always guarantees that the logical network has better results than the physical connections by improving the network configuration with Algorithm 4.

4.9 - Implementing a Cumulative Distribution Function

Typically, there are time intervals (*DataInt*) between the generation of packets at a node due to the random changes in the computation times due to irregular variations in the hardware components when processing information. We used a well-known method based on the cumulative distribution function (CDF) to generate values for the data interval between the generation of packets at the source nodes. CDF is a distribution function of the random variable *x*, defined as:

$$\mathrm{F(x) = P(X \leq x) = \sum t \leq xf(t)} \tag{4.2}$$

In our work, the data interval is generated as follows:

$$\mathrm{DataInt} = -\frac{1}{\lambda}\log(1 - U01()\,) \tag{4.3}$$

In the expression, `U01()` corresponds to an uniform random number generation algorithm, with an output k, where $0.0 \leq k \leq 1.0$. Notice the following algorithm that describes the data interval generation at the source as well as the `U01()` algorithm.

Algorithm 5: Algorithm to generate the data interval between the generation of packets at the source node.

```
lambda : float value correspondent that is equal to 0.5

define dataInt as ((-1.0/lambda) * Math.log10(EmulatorUtils.U01()))
```

```
method U01(){
   define v as 1.0
   while v is equal to 1.0 then
      define v as a random float k where 0.0 ≥ k < 1.0
   end
}
```

Thus, CDF will be used in every emulation of both emulator modes in order to induce those casual random variances in the technological processing.

4.10 - Emulating with Error Probability

Network coding was initially presented as a method for error correction and many authors have shown that it may improve the network protection against errors. We decided to implement an error probability option in the emulator. The user has the option of choosing the link error probability for each emulation, independently of the emulation mode. Assuming P_k as the packet loss probability and *s* as the number of jumps (hops) between links, the error probability, P_e, of a path, is given by the following equation:

$$Pe = 1 - (1 - P_k)^s \tag{4.4}$$

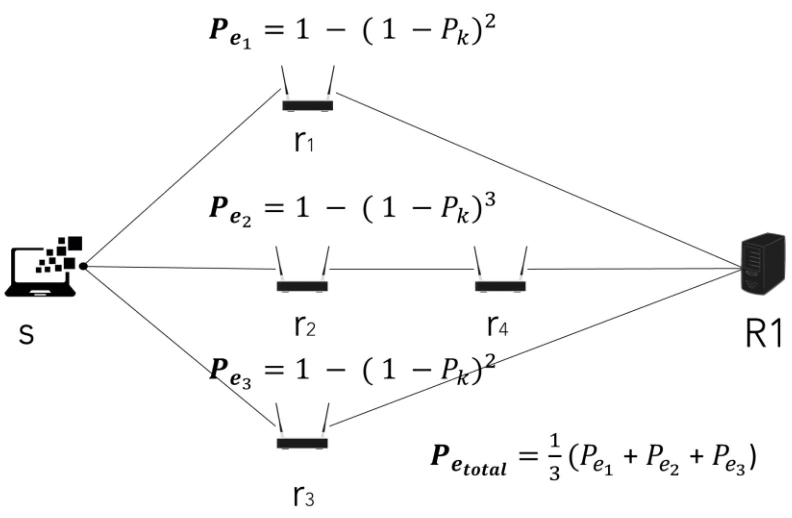
There are three classification options for the values of the packet loss probability, P_k:

- Null – there are no errors at the links, i.e., the error probability is 0;
- Low – the error probability at each link is 2%;
- High – the error probability at each link is 10%.

Each path error probability is calculated individually. The total error probability of a network with *r* path is calculated as follows:

$$P_{e_{total}} = \frac{1}{r}(P_{e_1} + \cdots + P_{e_r}) \tag{4.5}$$

Notice the following example of the calculation of a route error probability and the total error probability for the traditional method (no coding), as shown in Figure 4.10.



$\boldsymbol{P_{e_1}} = 1 - (1 - P_k)^2$
r1
$\boldsymbol{P_{e_2}} = 1 - (1 - P_k)^3$
s
r2
r4
R1
$\boldsymbol{P_{e_3}} = 1 - (1 - P_k)^2$
r3
$\boldsymbol{P_{e_{total}}} = \frac{1}{3}(P_{e_1} + P_{e_2} + P_{e_3})$

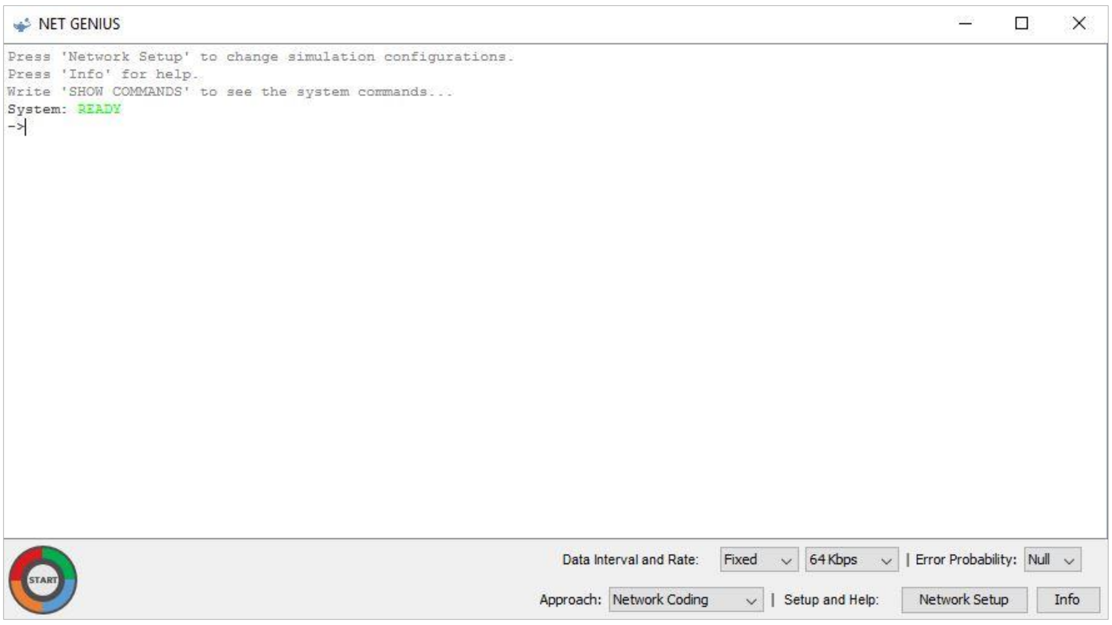
Figure 4.10 - Error probability example with three paths.

Depending on the error probability, P_e, there may be a packet loss which means that the packet that suffers that error is lost forever. In the traditional approach, when a packet is lost, the lost information cannot be recovered, unless the packet is retransmitted. This happens, because in the traditional approach there are only native packets. In the network coding approach, when a packet is lost, the information may be recovered by linear combinations, because the packets are coded together and typically each coded packet is a combination of several native packets.

4.11 - Console Options and Commands

In the setup window the user may choose what to emulate and how to emulate. In the console window the user may emulate the network and see the results, as well execute console commands. When the user starts an emulation, the emulation button will change by presenting to the user information about the emulation state.

The console offers some options to the user, as shown in Figure 4.11.



```
Press 'Network Setup' to change simulation configurations.
Press 'Info' for help.
Write 'SHOW COMMANDS' to see the system commands...
System: READY
->|
```

Data Interval and Rate: Fixed | 64 Kbps | Error Probability: Null

Approach: Network Coding | Setup and Help: Network Setup | Info

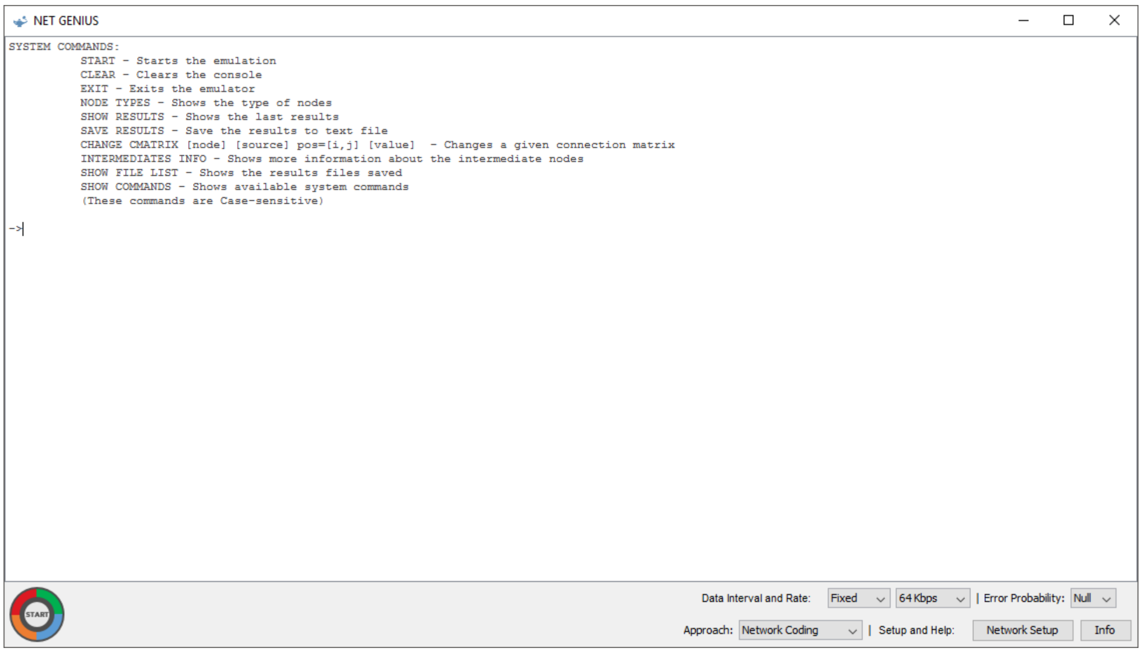
Figure 4.11 – Net Genius console v4.0.

The user may choose the data rate and data interval of the source nodes for each emulation. The data rate is equal to the amount of data produced by the source per unit of time and the data interval is the amount of time between the generation of data. The data rate can vary between 64 *kbps* and 1024 *kbps*. The data interval may be fixed or variable. The user may also choose the approach, if available, and the error probability for the emulation.

The user may see other information through the console commands. The console has the following console commands:

- SHOW COMMANDS – this command shows the available commands and its description;
- START – this command starts the emulation with the selected emulator mode as the Start Button;
- CLEAR – this command clears the emulator console;
- NODE TYPES – this commands describes the emulator nodes by showing the user a brief description of each type of node implemented;
- EXIT – by executing this command the console is closed and the user returns to the setup window, i.e., it exits the main window of the emulator;
- SHOW RESULTS – this command shows the results of the last emulation with the detailed information;

- SAVE RESULTS – this command saves the results of the emulation executed into a file that is readable by the common user;
- INTERMEDIATES INFO – this command shows the detailed information about the connections matrices of the intermediate nodes of the current network;
- SHOW FILE LIST – this command shows the list of files with emulation results;
- CHANGE CMATRIX [*node*] [*source*] pos=[*i*,*j*] [*value*] – this commands allows the user to change a connection matrix. The user needs to define the matrix he wants to change by setting up the intermediate node ID in *[node]* and the source node associated to the matrix in *[source]*. Then the user needs to define the matrix position we wants to change and the value in *[i,j]* and *[value]* respectively. By executing this command the emulator will show the new configuration. Note that changing the configuration does not always guarantee that the emulation will start, i.e., the user may change the configuration and the emulator may not be capable of emulating the network. Also, note that, in the advanced mode, the genius will improve the network in an independent manner, even if the user changes the connections matrices with this command. Thus, it is advised that this command should be only used using the basic mode.



```
SYSTEM COMMANDS:
          START - Starts the emulation
          CLEAR - Clears the console
          EXIT - Exits the emulator
          NODE TYPES - Shows the type of nodes
          SHOW RESULTS - Shows the last results
          SAVE RESULTS - Save the results to text file
          CHANGE CMATRIX [node] [source] pos=[i,j] [value]  - Changes a given connection matrix
          INTERMEDIATES INFO - Shows more information about the intermediate nodes
          SHOW FILE LIST - Shows the results files saved
          SHOW COMMANDS - Shows available system commands
          (These commands are Case-sensitive)

->
```

Data Interval and Rate: Fixed | 64 Kbps | Error Probability: Null
Approach: Network Coding | Setup and Help:

Figure 4.12 – Net Genius console of the final version

Notice that the commands are case sensitive, so must be written as presented. These console commands complete the console buttons and offer to the user a set of options that are very useful. Therefore, the final version of the emulator console offers more options and much more analysis metrics, as shown in Figure 4.12. At the end of each emulation, the console presents the results of the emulation. Also the user may access the help menu by clicking in “Info” button.

4.12 - Emulation Results

The emulator that was created allows one to observe the main differences between the traditional approaches and the network coding approaches by analysing the three QoS parameters presented earlier. The network configuration used for these emulations is the one presented in Figure 4.13, encompassing one source node, six intermediate nodes and one destination node. In our emulations CDF is always used the for the data interval between the generation of new packets at the source nodes. The source’s data rate values varies between 64kbps to 1024 kbps. Each link has a 54Mbps capacity and the packet size is 1000 bytes. The network coding approach uses the XOR operation to code and decode the packets.

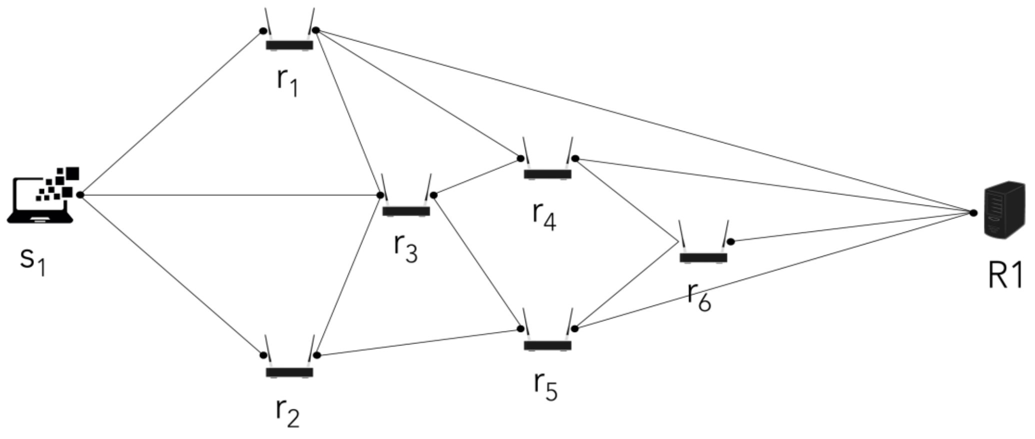


Figure 4.13 - Network topology used for the emulations.

The scenarios considered for our study and emulations are the following ones:

- Case 1: in this scenario the traditional method was used and no errors occur, i.e., all intermediate nodes are simply forwarding the packets to the next node and the emulator is working on basic mode.
- Case 2: in this scenario the traditional method was used and the error probability is low, which means that the error probability is 2% for each link. The emulator is also working on basic mode.
- Case 3: in scenario 3, the traditional method is used once more and the error probability is high (10% for each link). The emulator is also working on basic mode here.

- Case 4: in this scenario the emulator is also working in basic mode. The distributed network coding approach is used with no error probability.

- Case 5: in this scenario the emulator is once more working on basic mode and the distributed network coding approach is used but the error probability is low.

- Case 6: in this scenario the emulator is working on basic mode. The distributed network coding approach is used with a high error probability.

- Case 7: in this scenario, the emulator is working on advanced mode, i.e., the network is centralized and is genie-based. The error probability is zero and the network coding approach is used.

- Case 8: in this scenario the emulator is running on advanced mode and the error probability is low. The centralized network coding approach is used.

- Case 9: the centralized network coding approach is used in this scenario. The emulator is working on advanced mode with a high error probability.

Although there are quite a few parameters to study, this work focus on the three QoS parameters, namely the PLR, the latency and the jitter calculated by (3.3), (3.4), and (3.7), respectively. To run the emulations it was used a computer with 8 GB of RAM and having an Intel Core i7-2630QM 2.0 GHz processor. Notice that the emulations results are variable, i.e., two emulations of the same scenario seldom have the same results. These results vary due to irregular variances of the technological components and operating system when processing information. To make the results more robust, what is presented are the average results for the 135 possible scenarios (9 case studies, 5 data rate values and 3 PLR values), resulting from the performing of 30 emulations. The error margin, ρ, may be calculated via the expressions (4.5) and (4.6), where n is the number of emulations, $\bar{z}$ is the average result and z_i is the exact result for the *i* emulation:

$$\tau = \sqrt{\frac{\sum_1^n (z_i - \bar{z})^2}{n}} \tag{4.5}$$

$$\rho = \bar{z} \pm 1{,}96\tau \tag{4.6}$$

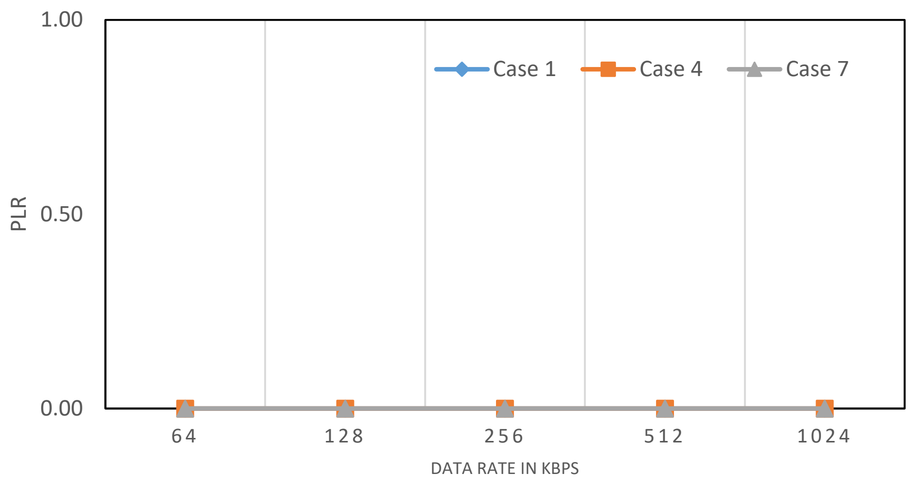


Figure 4.14 - PLR results for case 1, 4 and 7.

Figure 4.14 shows the emulation results of PLR for cases 1, 4 and 7. In these cases, the error probability is always zero, which means that no failures occurs on the links. It is clear that in all cases the PLR is always zero i.e., the destination node is capable of receiving all the transmitted data from the source node. These values were expected and there is no difference between cases 1, 4 and 7. Notice that case 7 is using different topology configurations in the emulations, which makes harder to analysE and compare the results between case 4 and case 7.

In terms of latency, there are some differences between cases, as shown in Figure 4.15. The best results are the ones from case 1, which means that the traditional approach reaches better latency values than the network coding approach when no failure occurs. The lowest latency value is 9.9 *ms* in case 1 and the highest value is 14.7 *ms* in case 7. The reason for the latency values in the traditional approach being lower than the latency values in the network coding approach is that the former spends less time processing the packets at the nodes than the latter, since in there is no need to code or decode the packets.

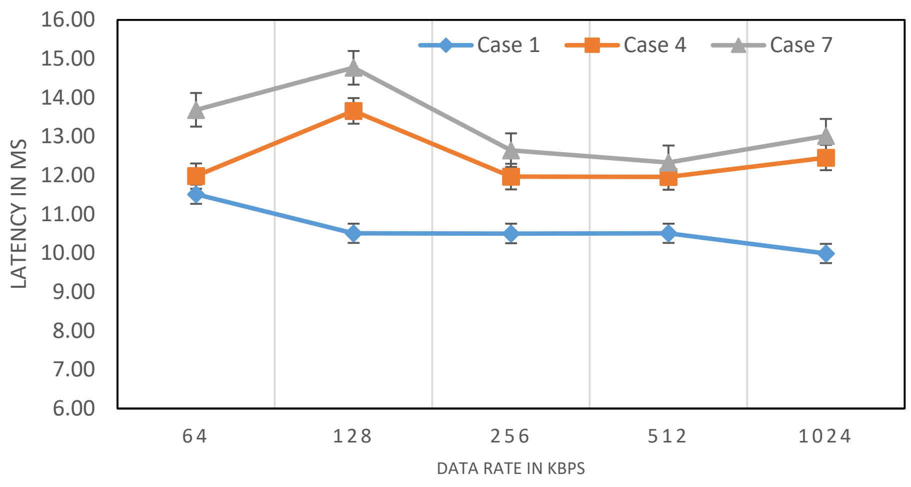


Figure 4.15 - Latency results for cases 1, 4 and 7.

Figure 4.16 shows the jitter results for cases 1, 4 and 7. As in the previous analyses, the traditional approach reaches the best results. The jitter values varies between 14 *ms* and 43 *ms*.

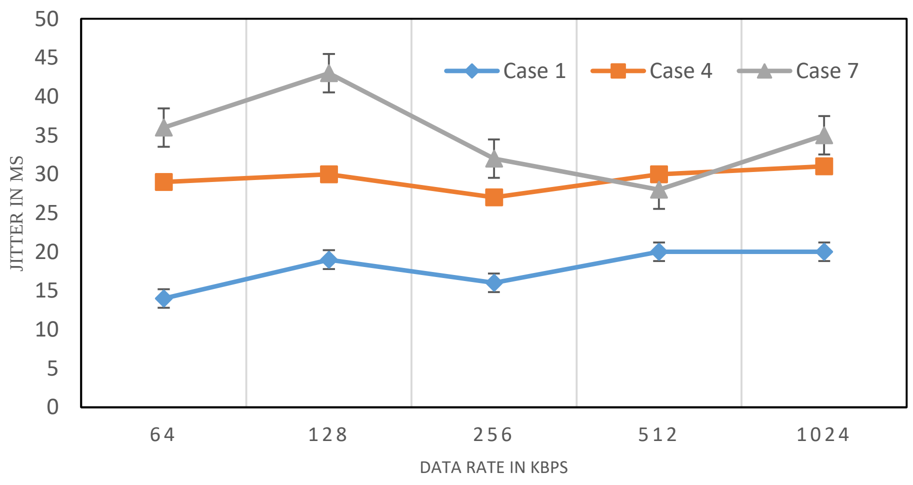


Figure 4.16 - Jitter results for cases 1,4 and 7.

One may also assert that the jitter increases when the data rate value increases, which happens consistently in the emulations. Similarly to what just seen to happen in respect to the latency, since the nodes working with the traditional approach do not need to code or decode packets, it is normal that the jitter values are lower than the jitter values from the network coding approaches.

Notice, that when a node is coding or decoding, the processing time is higher, because to code or decode a packet a node typically needs at least two packets that have different arrival times.

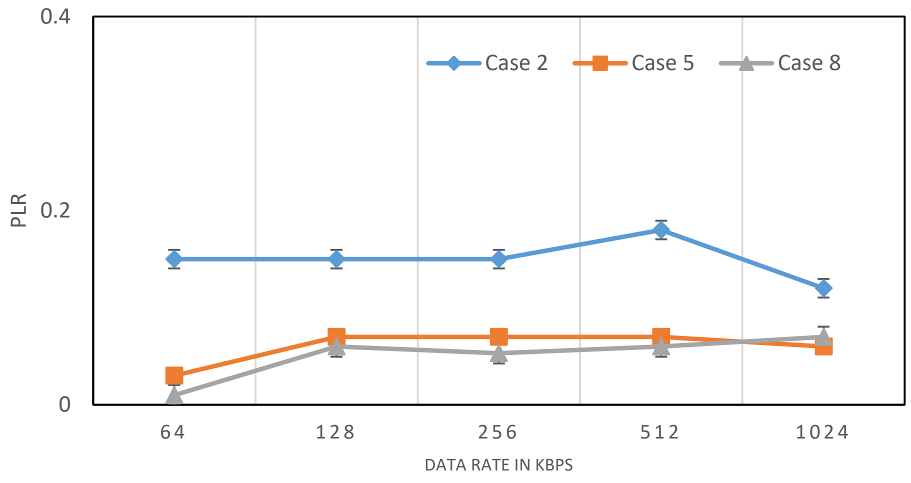


Figure 4.17 - PLR results for case 2, 5 and 8.

Figure 4.17 shows the PLR results for case 2, 5 and 8. In these cases, the error probability is low, i.e., each link has an error probability of 2%. The results show that the network coding approach of the genie-based network reaches the best PLR values, reaching the minimum PLR value of 1%. In case 2 the PLR values are significantly higher than case 2 and case 5, reaching the highest PLR value of 18%, which is a significant loss. The difference of values between case 5 and 8 are not that much, but it is clear to see that the genie-based network reaches better results in terms of PLR. In the three cases, the PLR tends to increase with the data rate increase.

In terms of latency, the results are bit different from the previous cases, as showning Figure 4.18. The lowest latency values are the ones from case 2, which means that the traditional method obtains better latency results than the network coding approach. Due to the coding and decoding processing time and bucket time, it was expected that the latency values were higher in the NC approaches. In the NC approaches and in the traditional approach each node spends time processing the packet header and buffering the packet. In NC, there is also the bucket time, because the node needs to wait in the bucket until the node get all the needed packets or until it the bucket reach the timeout. This bucket time increases the latency, which explains these results.

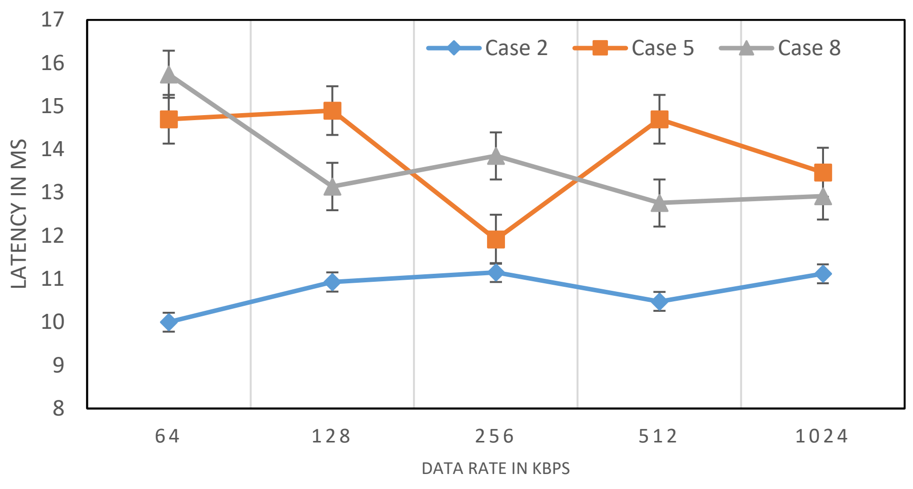


Figure 4.18 - Latency results for cases 2, 5 and 8.

Also, the results of case 2 are very similar to the results of case 1. In the previous cases, the genie-based network delivered worst results than the case of the distributed network, but in these situations the results are not the same. The latency values for case 8 are better than the latency values for case 5. Also the latency values of case 8 varies less than the values of case 5. Nevertheless, the highest value is 15,74 *ms* in case 8, as shown in the previous figure.

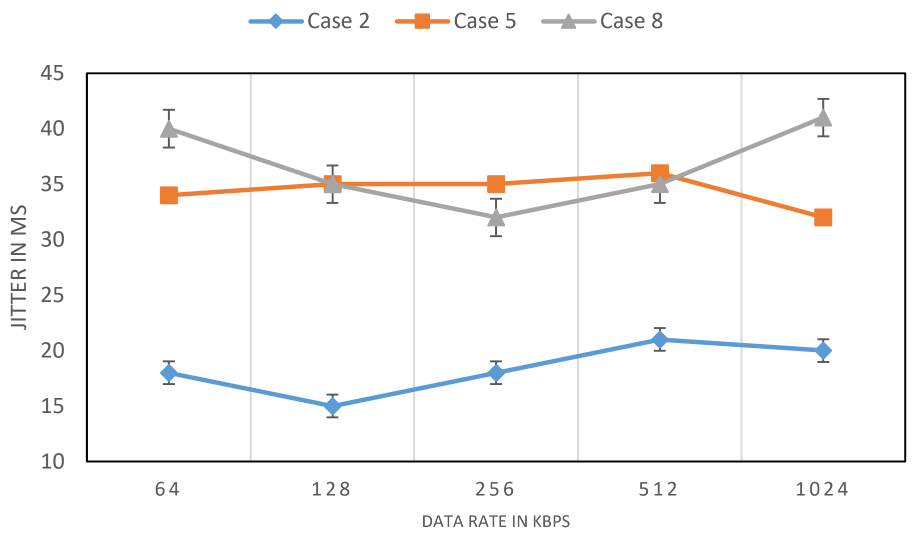


Figure 4.19 - Jitter results for case 2, 5 and 8.

Figure 4.19 shows the results for the jitter in cases 2, 5, and 8. Comparing the results with the previous cases, the jitter values are a slightly higher. Once again, the traditional method reaches the best jitter results, reaching the lowest value of 15 *ms*. The highest value is 41 *ms* for case 8. One can also note that there is not a high overall difference between cases 5 and 8, where the jitter values varies between 32 *ms* and 41 *ms*.

Figure 4.20 shows the PLR results for case 3, 6, and 9. In these cases, the error probability is high, i.e., each network link has an error probability of 10%. It is clear that case 9 attain a lower PLR value than the ones in cases 3 and 6, which means that the genie-based network is more resilient against failures than the distributed network coding approach and the traditional approach. Case 3 exhibits the worst results, reaching a PLR of 58%, which means that more than half the data transmitted is lost. Both network coding approaches stay under 35% of losses. In the traditional approach the losses are always higher than 48%. The difference of the traditional approach in case 3 to the network coding approach for cases 6 e 9 is significant and, once again, the second approach attains better results than the first when there failures occur.

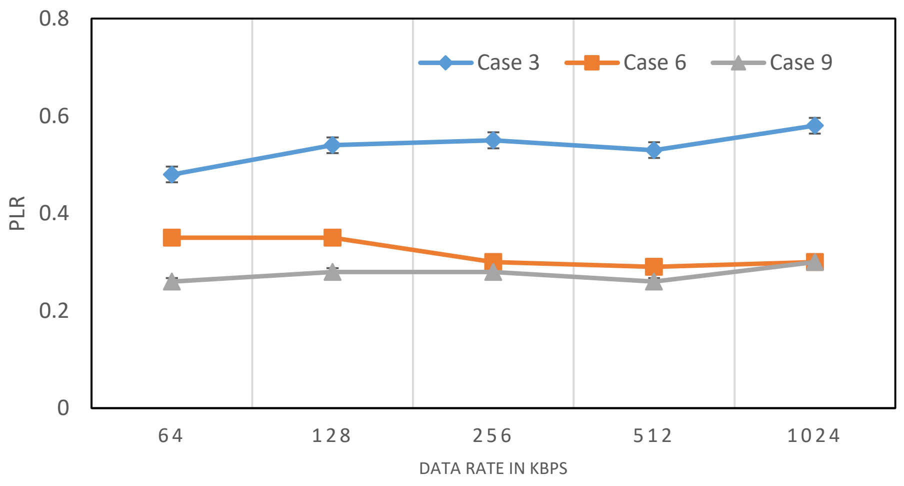


Figure 4.20 - PLR results for cases 3, 6 and 9.

Figure 4.21 shows the latency results for cases 3, 6 and 9. The results are similar to the ones of previous cases 2, 5, and 8, where the traditional approach reaches the best latency results. As expected, when the data rate increases, the latency decreases. The latency values of the network coding approach are higher than the traditional approach because the intermediate nodes and the destination nodes need to wait for slower packets, i.e., the nodes that are coding or decoding need to wait until they get all the needed packets or until they reach the bucket timeout. When the error probability is higher, there is a lot of packets that are lost, which means that the nodes that are coding and decoding will wait more time.

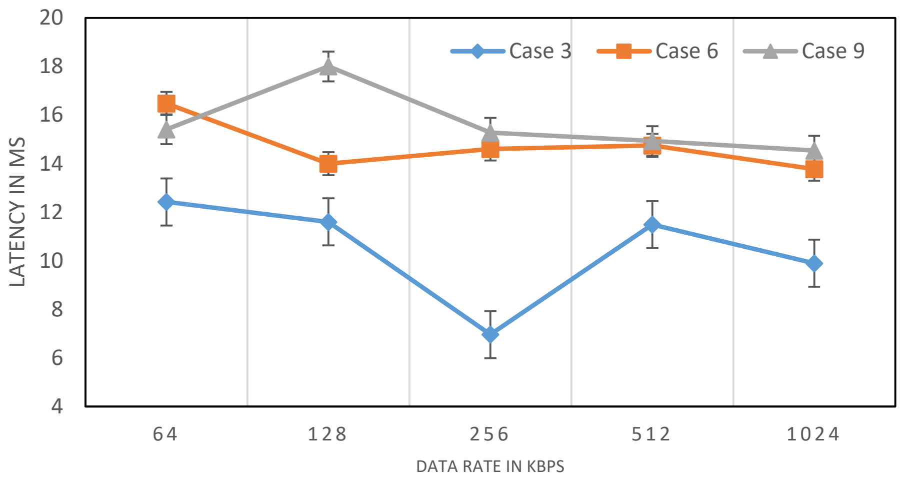


Figure 4.21 - Latency results for cases 3, 6 and 9.

Figure 4.22 shows the jitter results for cases 3, 6 and 9. The jitter values varies between 14 *ms* in case 3 and 45 *ms* in case 6. As in the previous cases, the jitter values for the traditional approach are better than the jitter values for the network coding approach. This happens, because the network coding approach uses a bucket timeout to wait for slower packets that never arrive due to failures. For all cases, when the data rate increases, the jitter tends to increase.

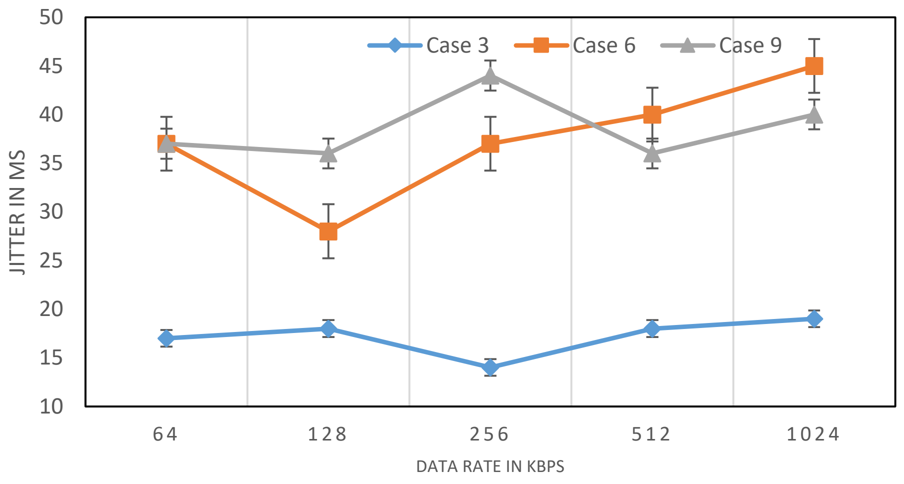


Figure 4.22 - Jitter results for cases 3, 6 and 9.

Analysing the results of the three QoS parameters for all cases 1-9 it is clear to see that network coding is more resilient to failures than the traditional approach. When the error probability is higher than zero, the traditional approach gets worse PLR results than NC. Because NC uses coded packets from the source, when some packets are loss, they still can be retrieved by linear combining other linear independent combinations of packets. In the traditional approach there are no coded packets, so when a packet is loss there is no way to recover that packet unless there is a retransmission of the lost data. When there are no errors in the network, the results of the traditional approach and the network coding approach are similar. Nevertheless, when one is sure that the network has no failure, then the traditional approach is preferable to NC because there is no need to spend resources on coding and decoding packets.

The NC approach is worse than the traditional approach in terms of latency and jitter, due to the processing time in the nodes. In NC the nodes are computing the packets before forwarding the same, so the processing time in the nodes is higher than the processing time in the traditional approach. Although NC is more resilient to failure than the traditional approach, this not means that is quicker. In order to decide which one is the best approach, one needs to exactly define what "best approach" means. If the best approach targets lowering the latency values, than one should opt for the traditional approach. This to say that while being more resilient, NC is not faster than the traditional approach.

In short, we may see that the results show that both network coding approaches are more resilient than the traditional approach in terms of protections against failures. Obviously, real networks without failures are rare to be found, and therefore this fact makes NC the best option for real networks. Even if there were no failures in the network, the best approach in terms of reliability and flexibility is always network coding, specifically the centralized network coding approach that improves the network cost. Furthermore, a genie-based network using network coding is more resilient than a distributed network using NC, because the genius is constantly monitoring and improving the network configuration.

Since the genius is improving the network, the network cost is also decreasing over the self-reconfigurations of the network that the genie dictates. In the emulations of cases 7-9 the genius has improved the network reaching a cost of 13 of 15, calculated from CRa in 4.1. In all the other cases the cost was always the maximum cost, 15 of 15, which means that by decreasing the cost, the genius still guarantees a resilient and robust network. Decreasing the network cost and simultaneously be able to increase the network resilience is very important to guarantee a better performance in terms of both PLR and network traffic.

Chapter 5

Conclusions

This chapter presents the conclusions of this dissertation and describes possible future directions for this work.

5.1 - Main Conclusions

This dissertation presents a network coding emulator that assesses the impact of having network coding over user-defined networks. The emulator was developed in order to create a tool that would help to analyse the behaviour of network coding and also to provide a simple and viable tool to test and analyse network coding independently of the network topology or type.

Linear network coding (described in Chapter 2) is the central concept to be assessed in this work. The implementations of network coding techniques lead to the use of extra information in the packet headers and operations over packets at the nodes. The previous results in [8] were essential to this work because they served as a starting point and allowed validate the emulator.

Despite the existence of other network tools, this work provided the first tool that flexibly emulates network coding in Java. Our emulator is capable of emulating various network topologies and different coding configurations. Emulations are very important to prove certain theories and the Net Genius may be helpful to the information theory community and other research communities.

The results presented in Chapter 4 allow us to conclude that network coding is a resilient technique for networks, independently of the existence or non-existence of errors or link failures. As shown by some authors (see Chapter 2), the network coding approach guarantees a more secure network because only combination of packets flow through the network, avoiding the routing of native packets at the nodes and therefore usually decreasing the number of packets circulating between nodes. This not only guarantees a more secure network but also ensures robustness in the routing of data. Furthermore, network coding increases the protections against failures as shown in the emulation results and it was also shown that a genie-based network using network coding achieves betters results due to the automatic network configuration improvement that not only improves the packet delivery ratio but also decreases the network cost.

Finally, in terms of latency and jitter metrics, NC leads to results that are worse than the ones obtained with the traditional approach. However, these disadvantages are compensated by a higher network throughput, a more resilient network, and lower network usage rate. It can be concluded that a network coding approach is the preferable choice for lossy networks due to its capability of preventing data losses due to the linear combinations made with LNC at the nodes, which answers the second research question presented in Chapter 1. In spite of its much higher implementation complexity in a real and large network, the centralised network coding approach is the one that achieves the best metrics in all the scenarios emulated in this work. In answer to the first research question, combining network coding with the generations mechanism and coding at the source is the best coding and decoding mechanism.

5.2 - Future Work

A natural extension to this work would be to include error correction mechanisms in the emulator. Another possible development to the improvement of the Net Genius would be the development of a more user friendly interface for the configuration of the network topology with a drag-and-drop tool.

Testing other scenarios, such as situations with node failures, would also be important to be studied. A more elaborate future extension would be the introduction of other agents such as the traditional eavesdropper (Eve), who tries to decode some information.

References

[1] N. Cai and R. W. Yeung, "Network coding and error correction," *IEEE Information Theory Workshop*, 20-25 October Bangalore, India, 2002.

[2] D. E. Lucani, J. Jespersen, M. Medard, F. H. P. Fitzek and B. Lindberg, "Chocolate Cloud," 2014. [Online]. Available: https://www.chocolate-cloud.cc/.

[3] J. Hansen, D. E. Lucani, J. Krigslund, M. Medard e F. H. P. Fitzek, "Network coded software defined networking: enabling 5G transmission and storage networks," *IEEE Communications Magazine*, vol. 53, nº 9, pp. 100-107, September 2015.

[4] D. Vukobratovic et al., "CONDENSE: A Reconfigurable Knowledge Acquisition Architecture for Future 5G IoT," *IEEE Access*, vol. 4, pp. 3360 - 3378, July 2016.

[5] M. A. Vazquez-Castro e T. Do-Duy, "Network Coding function virtualization," em *IEEE 17th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Edinburgh, UK, 2016.

[6] D. Vukobratovic, D. Jakovetic, V. Skachek, D. Bajovic e D. Sejdinovic, "Network function computation as a service in future 5G machine type communications," em *9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC),* Brest, France, 2016.

[7] K. A. Agha, Network Coding, John Wiley & Sons, Inc, 2012.

[8] B. Saeed, P. Rengaraju, C. Lung, T. Kunz and A. Srinivasan, "QoS and protection of wireless relay nodes failure using network coding," *in Proceedings of Network Coding (NetCod),* pp. 1-5, 2011.

[9] A. Hevner and S. Chatterjee, "Design science research in information Systems," *Springer,* 2010.

[10] H. A. Simon, The science of design: creating the artificial in The Sciences of the Artificial, MIT Press, Cambridge, MA, 1996, pp. 111-138.

[11] M. Médard and A. Springtson, Network Coding: Fundamentals and Applications, Waltham, MA: Academic Press (Elsevier), 2011.

[12] R. Ahlswede, N. Cai and S.-Y. R. Li, “Network information flow,” *IEEE Transactions on Information Theory*, vol. 46, pp. 1204 - 1216, 2000.

[13] C. Fragouli and E. Soljanin, "Introduction," in *Network Coding Fundamentals*, Now Publishers, 2007, pp. 1-9.

[14] T. Ho and D. S. Lun, Network Coding: An Introduction, Cambridge University Press, 2008.

[15] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard and J. Crowcroft, “XORs in the air: practical wireless network coding,” *in Proc. of SIGCOMM Comput. Communication*, vol. 36, pp. 243-254, October 2006.

[16] J. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher and J. Barros, "Network coding meets TCP," *in Proceedings of IEEE INFOCOM*, pp. 280 - 288, Rio de Janeiro, Brazil, April 2009.

[17] H. Seferoglu, A. Markopoulou and K. K. Ramakrishnan, "I2NC: Intra- and Inter-session Network Xoding for Unicast Flows in Wireless Networks," *Proc. of IEEE Conf. on Computer Commun. (INFOCOM)*, April 2011.

[18] C. Fragouli, J. L. Boudec and J. Widmer, “Network coding: an instant primer,” *in Proc. of SIGCOMM Comput. Commun.*, vol. 36, pp. 36-38, January 2006.

[19] A. G. Dimakis, B. P. Godfrey, Y. Wu, M. O. Wainwright and K. Ramchandran, “Network Coding for Distributed Storage Systems,” *in Proceedings of IEEE Transactions on Information Theory* , pp. 4539 - 4551, August 2010.

[20] C. Hollanti, D. Karpuk, A. Barreal and H.-f. F. Lu, "Space-time storage codes for wireless distributed storage systems," *in Proceedings of Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE)*, May 2014.

[21] S. Li and A. Ramamoorthy, "Protection against link errors and failures using network coding," *IEEE Transactions on Communications*, vol. 59, pp. 518 - 528, 2011.

[22] S. Lin e D. Costello, Error Control Coding: Fundamentals and Applications, Prentice Hall, 2004.

[23] A. E. Kamal and A. Ramamoorthy, "Overlay protection against link failures using network coding," *in Proceedings of the Conference on Information Sciences and Systems (CISS),* 2008.

[24] S. Ramamurthy and B. Mukherjee, "Survivable WDM mesh networks. Part I - Protection," *Proc. IEEE INFOCOM,* vol. 2, pp. 744-751, 1999.

[25] R. Chang, S.-J. Lin and W.-H. Chung, "Transmission protocol design for binary physical network coded multi-way relay networks," *Vehicular Technology Conference (VTC Spring),* no. 79 IEEE, pp. 1-5, 18-21 May, 2014.

[26] E. Biglieri and G. Taricco, Transmission and Reception with Multiple Antennas: Theorectical Foundations, Hanover, Massachusetts: USA: Now Publishers, 2004.

[27] B. Nazer and M. Gastpar, "Computing over multiple-access channels with connections to wireless network coding," *Proc. of IEEE int. Symp. on Inf. Theory, Seattle, USA,* pp. 1354-1358, July 2006.

[28] A. Tarable, I. Chatzigeorgiou and J. Wassell, "Randomly select and forward: erasure probability analysis of a probabilistic relay channel model," in *IEEE Information Theory Workshop*, Taormina, 2009.

[29] J. Goseling, M. Gastpar and J. Weber, "Random access with physical-layer network coding," *Information Theory and Applications Workshop (ITA),* pp. 1-7, 10-15 February, 2013.

[30] E. Paolini, C. Stefanovi´c, G. Liva and P. Popovski, "Coded random access: applying codes on graphs to design random access protocols," *IEEE Communications Magazine,* pp. 144-150, June 2015.

[31] N.Cai and R. Yeung, "Secure network coding," *in Proc. IEEE International Symposium on Information Theory,* Lausanne, Jun 2002.

[32] A. Kalantari, G. Zheng, Z. Gao, Z. Han and B. Ottersten, "Secrecy analysis on network coding in bidirectional multibeam satellite communications," *IEEE Transactions on Information Forensics and Security,* vol. 10, pp. 1862 - 1874, May 2015.

[33] A. S. Khan, A. Tassi and I. Chatzigeorgiou, "Rethinking the intercept probability of random linear network coding," *IEEE Communications Letters,* vol. 19, no. 10, pp. 1762-1765, October 2015.

[34] X. Wang, W. Guo, Y. Yang and B. Wang, "A secure broadcasting scheme with network coding," *IEEE Communications Letters,* vol. 17, no. 7, p. 1435– 1438, July 2013.

[35] D. Silva, K. F.R. and K. R., "A rank-metric approach to error control in random network coding," *IEEE Transactions on Information Theory,* vol. 54, pp. 3951 - 3967, 2008.

[36] J. Lipčák, “Computation-with-finite-fields 1.0,” 10 2014. [Online]. Available: github.com/XLipcak/Computation-with-finite-fields.

[37] S. Abdelwahab, B. Hamdaoui, M. Guizani and T. Znati, "Network function virtualization in 5G," *IEEE Communications Magazine,* vol. 54, pp. 84 - 91, April, 2016.

[38] J. Heide, M. V. Pedersen, F. H. Fitzek and M. Médard, "On code parameters and coding vector representation for practical RLNC," *IEE International Conference on Communications (ICC),* pp. 1 - 5, Kyoto, Japan, June 2011.

[39] C. J. Colbourn, “Combinatorial aspects of covering arrays,” *Le Matematiche (Catania),* vol. 58, pp. 121-167, , vol. 58, pp. 121–167, 2004. 2004.

[40] A. Jones, I. Chatzigeorgiou and A. Tassi, "Binary systematic network coding for progressive packet decoding," in *Proc. of IEEE Inter. Conf. on Communications (ICC),* London, UK, 2015.

[41] B. Nazer and M. Gastpar, “Compute-and-Forward: Harnessing Interference Through Structured Codes,” *in Proceedings of IEEE Transactions on Information Theory* , pp. 6463 - 6486, October 2011.