UNIVERSITAT POLITÈCNICA DE CATALUNYA

BARCELONA**TECH**

School of Telecommunications Engineering of Barcelona

Simulation algorithms for *Network Coding*

(Final Project)

Guilherme Pinto Cardoso Alegria Quintela

September 2010

To my parents Divo and Luísa,
brother Gustavo and all of my
family and closest friends.

Acknowledgments:

This work would not have been possible without the support of my family, in particular my parents and brother, which words cannot express how truthfully grate I am to them for all that I am today.

I would like to thank my supervisor, Prof. Marcel Fernández (UPC-ENTEL), for all his support, guidance and assistance throughout the project.

I am also very much thankful to Prof. Vítor Silva (UC-IT) for his support, guidance and mainly for giving me the opportunity of developing my final project abroad.

I also want to thank Alberto J. González (UPC) for all his assistance during the initial stages of this project presenting me to P2P streaming with *Network Coding*.

I am grateful to Hamed Firooz (UWEE-FunLab) for his help understanding the NC OPNET tutorial and not forgetting also the OPNET Google Groups users for their prompt replies in moments of need.

In last but not least, I would like to thank my closest friends for their love, support and incessant encouragement during the past months.

Abstract

Network Coding is a new research field which has been growing its interest among Information Theory and Network Engineering experts for its ability of improving network throughput and robustness enhancement.

This thesis intends to explore and study Network Coding characteristics in order to develop a reliable simulation study on a specific network. Simulations were implemented within OPNET simulator interfaced with MATLAB.

The Linear Information Flow algorithm is also studied and implemented in order to use the resulting network codes within the network implemented above.

Resumen

Network Coding es una nueva área de investigación que ha incrementado su interés entre los especialistas en Teoría de la Información y Ingeniería de redes por su capacidad en mejorar el throughput de una red y su robustez.

Este proyecto pretende explorar y estudiar las características de *Network Coding* con el objectivo de desarrollar y implementar una simulación de una red específica. Las simulaciones se han realizado con el simulador OPNET interconectado con MATLAB.

El algoritmo de *Linear Information Flow* también fue estudiado y implementado con el intuito de utilizar los network codes generados en la red implementada anteriormente.

Resum

Network Coding és una nova àrea d'investigació que ha incrementat el seu interès entre els especialistes en Teoria de la Informació i Enginyeria de xarxes per la seva capacitat en millorar el throughput d'una xarxa i la seva robustesa.

Aquest projecte pretén explorar i estudiar les característiques de *Network Coding* amb l'objectiu de desenvolupar i implementar una simulació d'una xarxa específica. Les simulacions s'han realitzat amb el simulador OPNET interconnectat amb MATLAB.

L'algorisme de *Linear Information Flow* també va ser estudiat i implementat amb el intuit d'utilitzar els network codes generats a la xarxa implementada anteriorment.

Resumo

Network Coding é uma nova área de investigação que tem vindo a aumentar o seu interesse entre os especialistas em Teoria da Informação e Engenharia de Redes pela sua capacidade de melhorar o throughput de uma rede e sua robustez.

Esta tese pretende explorar e estudar as características de *Network Coding* com o objectivo de desenvolver e implementar uma simulação de uma rede específica. As simulações foram executadas no simulador OPNET interligado com MATLAB.

O algoritmo de *Linear Information Flow* também é estudado e implementado, com o intuito de utilizar os *network codes* gerados dentro da rede implementada anteriormente.

Contents

List of figures

List of tables

List of algorithms

Notation

NC	*Network Coding*
$\mathbb{F}$	*Finite Field*
GF	*Galois Field*
LNC	*Linear Network Coding*
RNC	*Random Network Coding*
LIF	*Linear Information Flow*
P2P	*Peer-to-Peer*
LAN	*Local Area Network*
FSM	*Finite State Machine*
UDP	*User Datagram Protocol*
TCP	*Transmission Control Protocol*
$[0^{i-1}, 1, 0^{h-1}]$	*a h-length vector with a 1 in the ith location and 0 otherwise*
$a_t(c)$	*a vector with the property that* $x.a_t(c) \neq 0$ *if and only if x is linearly independent of* $B_t \setminus \{b(c)\}$ *for some* $c \in C_t$
$b(e) \in \mathbb{F}^h$	*global coding vector for edge* $e \in E$
B_t	*the set of global coding vectors for sink t,* $B_t = \{ b(c): c \in C_t \}$
C_t	*h edges on edge-disjoint paths from s to t*
$C(e)$	*the capacity of edge* $e \in E$

E	*the set of edges*
$e \in E$	*an edge*
$\{e_{1,\ldots,} e_h\}$	*input edges connecting s' with s*
f^t	*a flow of magnitude h from s to t represented by h edge disjoint paths*
$f_{\leftarrow}^{t}(e)$	*predecessor edge of e on a path from s to t* $\in T$
$G=(V,E)$	the graph
h	*the smallest maximum flow from s to some sink t* $\in T$
$\Gamma_I(v)$	*set of incoming edges in node v*
$\Gamma_O(v)$	*set of outgoing edges in node v*
m_e	*the local coding vector for edge e* $\in$ *E, i.e.,* $m_e(e')$ *is the coefficient multiplied with y(e´) to contribute to y(e)*
$P(e)$	*the predecessor edges of e in some flow path*$\{f_{\leftarrow}^{t}(e) : t \in T\}$
s	*source node*
s'	*artificial node*
$start(e)$	*the node where edge e* $\in E$ *departs*
$T \subseteq V$	*the set of sink nodes*
$T(e) \subseteq T$	*the sinks supplied through edge e* $\in E$*, i.e.,* $T(e)= \{ t \in T: e \in f^t \}$
$t \in T$	*a sink node*
V	*the set of nodes*
$v \in V$	*a node*
$y(e)$	*the symbol carried by edge e*

1- Introduction

Nowadays, practical communication networks such as phone, internet, peer-to-peer, wireless and ad-hoc networks, represent more and more an indispensable role in our lives. Traditionally, in order to exchange data between nodes, communication networks use routing algorithms (*store-and-forward*), where a node receives a packet or symbol and consequently forwards it to the next node, in such a manner that packets travel through the network remaining intact between the source and the destination/receiver.

1.1 *Network Coding*

Network coding is a recent research field in information theory that changes the traditional idea of routing, instead of simply forwarding data or packets, intermediate nodes may also be used in order to recombine several input packets into one or more output packets with the objective of attaining maximum information flow in the network.

This method was introduced by R. Ahlswede *et al.* [1] in the beginning of the new millennium, demonstrating how network coding can increase the throughput of a network and enhance its robustness. The idea of this method is relatively simple: the intermediate nodes recombine the received packets, sending a linear combination of them and forming a new representation of the original packet, that in order to decode it and retrieve the original message in such a way that the receiver just needs to have a sufficient number of linear independent combinations of those packets in order to decode it and retrieve the original message, independently of which specific combinations it received.

The topics of network coding involve mainly networking, graph theory, algorithms of optimization, information theory, security, load balancing, *ad-hoc* networks and network monitoring.

A relevant number of research studies were reported since [1] resulting in a continuous advance of understanding *Network Coding*, growing each year the number of articles published concerning this field, as illustrated in Figure 1.1.

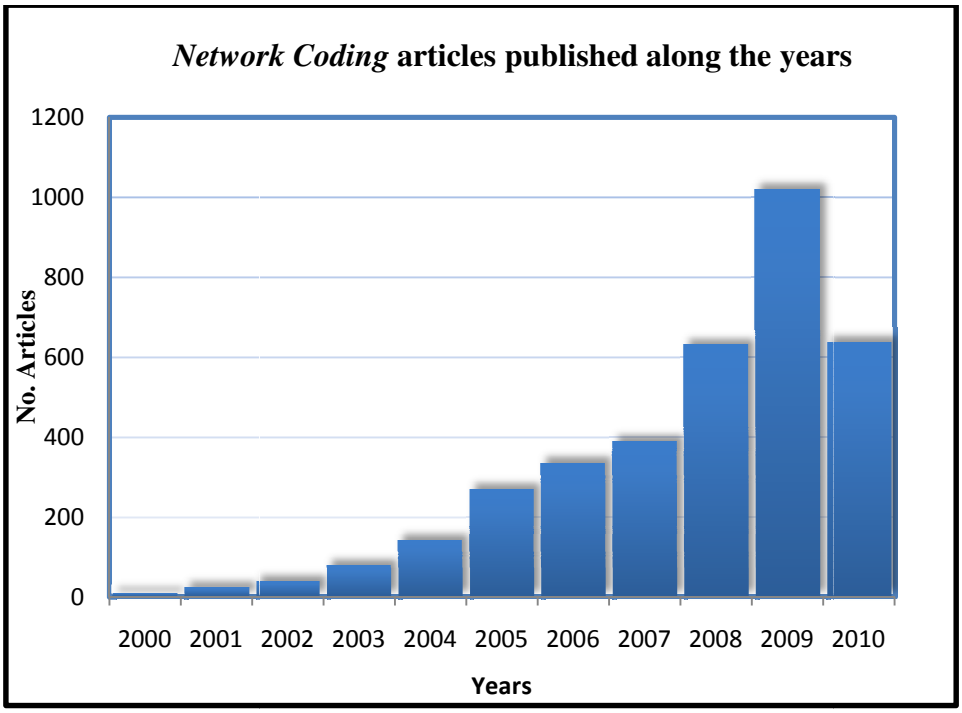

Figure 1.1 – NC articles published along the years. Data retrieved on July/2010 [22].

1.2 *Objectives*

The main objectives of this thesis are to study and explore all of the main *Network Coding* principals and characteristics in order to develop a reliable simulation of it within OPNET network simulator, implementing the Linear Information Flow algorithm [15] in MATLAB, introducing it on the network simulation developed before with OPNET.

1.3 *Thesis outline*

Chapter 2 covers all of *Network Coding* background research and properties studied.

The simulation models explored and used to simulate the studied algorithms and networks are described in Chapter 3.

Chapter 4 combines the last chapters with an intensive study of the *Linear Information Flow* algorithm, firstly with a theoretical approach and then its implementation and simulation using MATLAB and OPNET interfaces.

Finally, in chapter 5 the conclusion of this thesis is drawn along with some future research topics.

2 Background

This chapter provides a background on multicasting using *Network Coding* (NC) and reviews its major properties and developed researches, since it was first purposed [1], that were studied . Also will be taken in concern the many applications that have been associated to NC, for which *store-**code**-forward* brings relevant benefits.

2.1 *Network Coding* overview

Several benefits of using NC among communication networks have been proven in diverse subjects, such as throughput [1], end-to-end delay [2], wireless resources [10], security issues and robustness [3]. In this thesis, throughput benefits when multicasting are the main concern.

2.1.1 The butterfly network example

Assuming a communication network as a directed graph $G = (V, E)$, where V and E are the sets of vertices (nodes) and edges, respectively, consider the network topology represented in Figure 2.1.

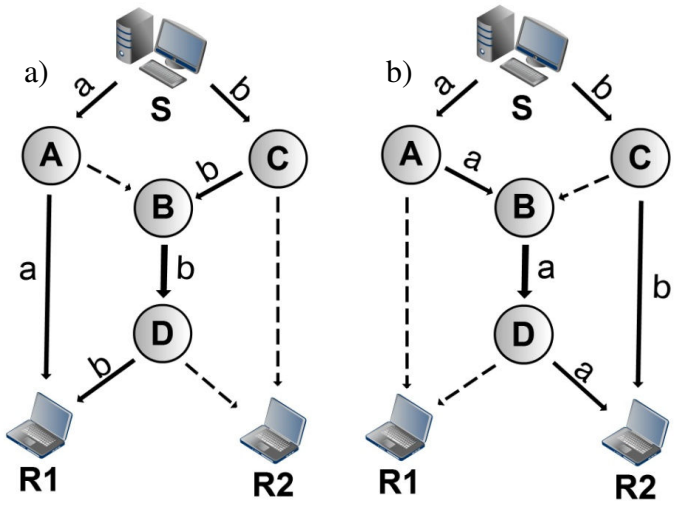

Figure 2.1 - Butterfly network with traditional routing a) node B only forwards symbol *b*. b) node B only forwards symbol *a*.

The butterfly network, represented in Figure 2.1 and Figure 2.2, is a classical example when demonstrating how NC can achieve throughput gain in a communication network. We have a Source, *s*, multicasting both bits *a* and *b* to receivers R_1 and R_2 in each time slot, so every channel/edge has unit capacity.

Using traditional routing, receiver R_1 would receive both bits *a* and *b* using all the network resources, where source S could route bit *a* along the path $\{AR_1\}$ and bit *b* along the path $\{CB, BD, DR_1\}$. Receiver R_2 could also receive both bits from *s*, receiving bit *b* through path $\{CR_2\}$ and bit *a* from path $\{AB, BD, DR_2\}$, as illustrated in Figure 2.1.

Now let's assume multicasting, in other words, let's consider that both receivers want to receive from *s* both bits *a* and *b*, simultaneously. If routers/nodes only forward the information they receive (traditional *store-and-forward*) the middle edge {*BD*} will be a bottleneck, which can either forward *a* or *b* to both receivers. If for example the bit forwarded by node B was bit *a,* receiver R2 would receive both bits *a* and *b* while receiver R1 would only receive bit *a*, as shown in Figure 2.1 b.

With NC both bits sent by S can be *XOR*ed (or linearly combined) at intermediate node B, as shown in Figure 2.2.

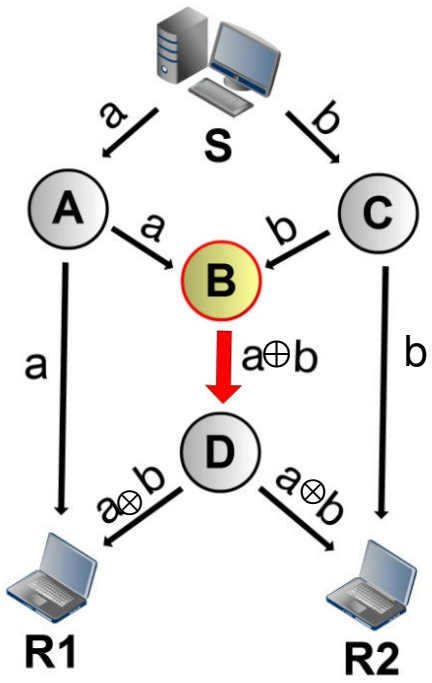

Figure 2.2 - Butterfly network using Network Coding.

A new packet, *c,* is created at node B, where $c = a \oplus b$, which is then sent through edge BD. R1 and R2 receive, respectively, {a, a⊕b} and {b, a⊕b}, being both able to retrieve *a* and *b* by solving a simple system of equations. The symbol ⊕ denotes the logical operation *xor* that is specified in table 2.1

⊕	0	1
0	0	1
1	1	0

Table 2.1 - XOR logical operations.

As shown, allowing the intermediate node in the network to codify (combine) the received packets and consecutively forward the new packet created to both receivers, achieves an increase of network's throughput when multicasting since multicast traffic network capacity can be achieved.

2.1.2 Main Theorem

When unicasting, only one receiver uses the network resources at a time, and the conditions provided to ensure the support of the network are widely known for a long time, while for multicasting, where we have to ensure that the same information is transmitted simultaneously to the multiple receivers within the network, those conditions have to be taken in concern so that the network supports the multicast at a certain rate.

NC Main Theorem [3], proved by Ahlswede *et al.* [1], facts that the conditions necessary to guarantee multicast at a certain rate to each receiver are the same as unicast since the intermediate nodes in the network are able to combine each information received.

2.1.2.1 Min-Cut Max-Flow

The Min-Cut Max-Flow Theorem was first proven by Menger [23], being also demonstrated afterwardsby others, such as Ford-Fulkerson [22]. In this thesis, it will be stated the Theorem having by reference [3].

Consider *G* a directed graph, *G= (V, E)*, with a set of vertices *V* and a set of edges *E*. Assuming that each edge *E* has unit capacity and allows parallel edges, let's consider a source (node) $S \in V$ which wants to send information to a receiver (node) $R \in V$.

Defining a *cut,* between *S* and *R*, as a set of edges which cannot be removed without disconnecting *S* from *R,* and its smallest value (the sum of the capacities of the edges in that cut) as *min-cut,* for unit capacity edges, the number of edges in the cut equals its value. The basic conclusion from the Min-Cut Max-Flow theorem is that the maximum information rate sent from *S* to *R* is equal to the min-cut value.

Having defined the min-cut, the Theorem states that if the min-cut value equals h, the maximum rate of the information sent by *S* to *R*, will be of h. Therefore, there will exist h edge-disjoint paths between source *S* and *R*, as shown in Figure 2.3.

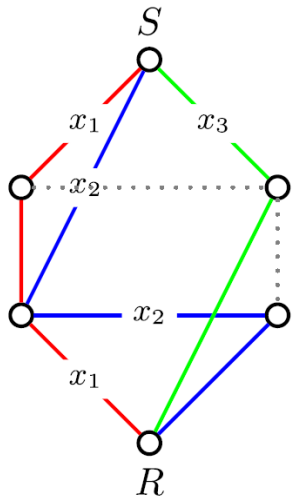

Figure 2.3 - Unicast connection in a network with unit capacity edges [3].

From the network topology defined in Figure 2.3 we can easily conclude that the min-cut value from this network equals three, existing three edge-disjoint paths between *S* and *R* that deliver symbols x_1, x_2 and x_3 to the receiver, *R*.

2.1.3 Linear Network Coding

It has been shown that intermediate nodes in a network, when performing NC, are able to combine a number of packets received into one or more new outgoing packets. Linear Network Coding (LNC) [2, 21], permits, instead of binary field operations, moving to larger field sizes, being able to perform more complex operations when combining incoming packets in intermediate nodes, becoming one of the most successful network coding algorithms as it permits achieving network capacity when multicasting, with relatively low complexity. In LNC each data unit is processed using finite fields $\mathbb{F}_q$ with q a prime number or, considering a Galois Field (GF), q= 2^m for some integer m, where $\mathbb{F}_{2^m}$ refers to $[0,\ 2^{m-1}]$.

Having a graph *G=(V,E),* where *V* represent nodes and *E* the edges through where data is transmitted, the butterfly network is once again the chosen example to best understand LNC operations within a network, illustrated in Figure 2.4.

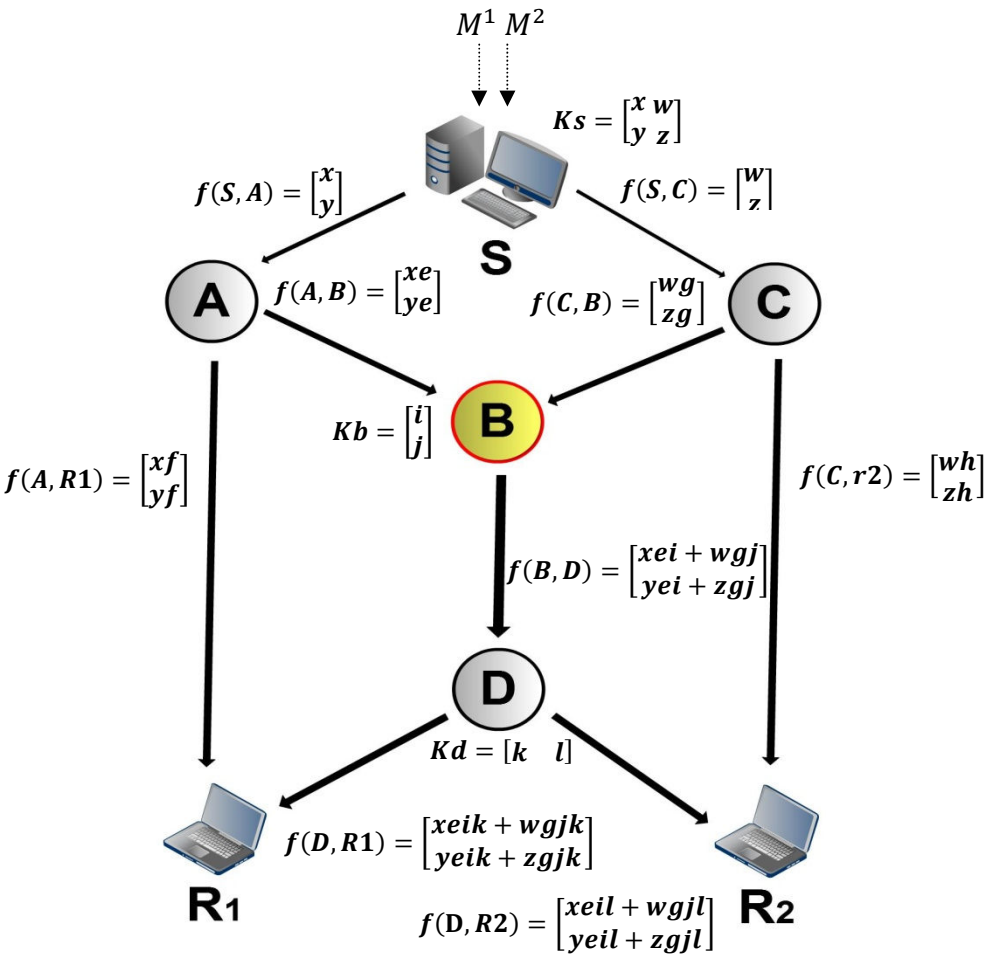

Figure 2.4 – LNC: Local/ global coding vectors of 2-dimensional linear network code.

2.1.3.1 Encoding

As shown in Figure 2.4, we have M^1, … , M^n original packets generated by source *S,* where n=2 in the previous example (2-dimensional linear network code). Each packet is associated to a sequence of *encoding vectors* g_1, … , g_n in field $\mathbb{F}_{2^m}$ and equals to an *information vector*,

$$X = \sum_{i=1}^{n} g_i M^i \tag{2.1}$$

being the sum executed (over the finite field $\mathbb{F}_{2^m}$) in each symbol position so,

$$X_k = \sum_{i=1}^{n} g_i M_k^i \tag{2.2}$$

where X_k and M_k^i is the *k*th symbol of *X* and M^i.

Updating Figure 2.4, the field associated is $\mathbb{F}_2 = \{0, 1\}$ being M^1= a and M^2= b, consecutively we have the result in Figure 2.5.

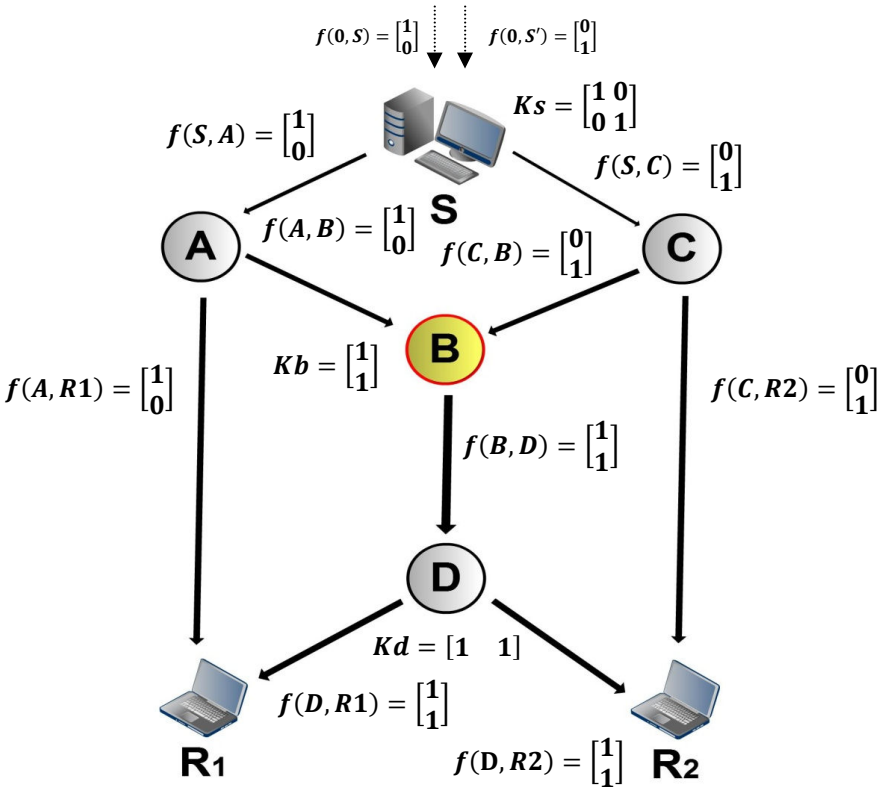

Figure 2.5 - LNC: local/ global coding vectors updated.

Concluding, with LNC, addition and multiplication are performed over the finite field $\mathbb{F}_{2^m}$, where the resulting packets are linear combinations of the original ones, still with the same length as the original packets.

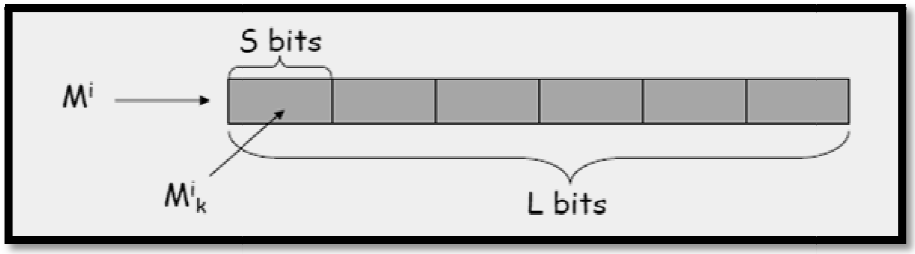

Figure 2.6 - NC packet format. Adapted from [25].

In Figure 2.6 it is illustrated a NC packet format where L stands for the length of each packet and S for S bits of M_k^i.

2.1.3.2 Decoding

At LNC, the original packets (M^1, … , M^n) are retrieved at the receiver nodes by executing a simple Gaussian elimination (decoding).

The encoded packet, as seen before, carries two kinds of information: *encoding vectors* $g_1,\ldots,g_n$ and *information vector X*. When a node receives (receiver) *m* number of *X*s (m ≥ n), it is able to decode them and retrieve successfully the original packets. Therefore the rank of the matrix (2.3) has to be *n*, in other words, the vectors belonging to (2.3) have to be linearly independent.

$$\begin{bmatrix} g_1,\ldots,g_n \\ g_1,\ldots,g_n \\ \ldots\ldots\ldots\ldots \\ g_1,\ldots,g_n \end{bmatrix} \tag{2.3}$$

LNC characteristics will be better focused in the following chapters, where algorithm implementations and simulations take place.

2.1.4 Random Network Coding

Random Network Coding (RNC) was first purposed by T. Ho *et al.* [26]. The main difference between RNC and LNC is the random selection of the linear combinations each node performs, being the selection over the field $\mathbb{F}_{2^m}$.

With RNC the probability of selecting linearly dependent combinations, as shown by T. Ho *et al.*, is at least $\left(1 - {}^{d}/_{q}\right)^{v}$ for q > d, where d is the number of receivers and v is the maximum number of edges receiving packets with independent randomized coefficients in any set of edges constituting a solution path from all sources to any receiver, [26]. Like LNC, as receiver nodes receive the many independent linear combinations as many as the number of sources, they can decode the message received.

2.1.5 Polynomial Time Algorithms

Alternatively to RNC and LNC, using of deterministic algorithms to design network codes is also a possibility.

S. Jaggi *et al*. in [24] describe a Linear Information Flow (LIF) algorithm, which will be implemented and reviewed in the following chapters.

2.2 *Network Coding Applications*

In this section, some of the applications where NC can have a significant impact are briefly described.

2.2.1 Ad-Hoc networks

The literal meaning of *ad hoc* in Latin is "for this", "for this purpose", so an ad-hoc network is a local area network (LAN) that is built as devices connect to each other. Mainly wireless, mobile, and P2P ad-hoc networks may be improved using NC.

2.2.1.1 P2P file distribution

The most famous method of this type would be BitTorrent, where each *peer* is capable of analyzing and transmitting any type of file over a network, using the BitTorrent protocol. A *peer* is any computer running an instance of a client.

The interest in this thesis in *peer-to-peer* (P2P) networks using NC was first due to the collaboration from A.J. González [24], which would have a further interest applied to the implementations and simulations developed in this thesis, described in the next chapters.

In networks such as P2P streaming networks this mechanism, as others described before, allows a throughput increase. Considering a P2P streaming with shared download, as illustrated in Figure 2.7, sending a linear combination of the different blocks of the original file that the *peer* wishes to send into the network, permits a throughput increase. The redundancy of the available information between *peers* enhances the use of NC characteristics, increasing the number of possible linear combinations of packets, solving the problem of locating the last missing blocks to complete the download.

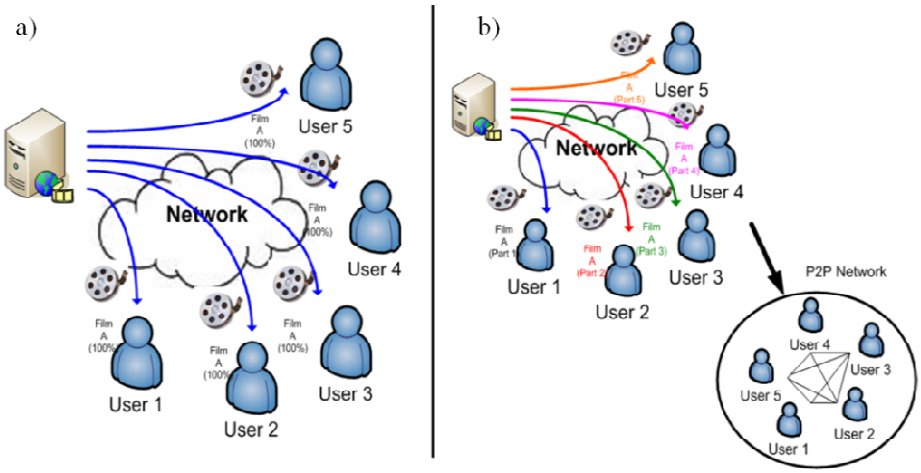

Figure 2.7 - a) Complete download - every user downloads the full file.
b) Shared download - every user download an equal share of the total of the file [24].

As the efficiency of the network increases, it reduces the probability of losing packet information and also the load which supports the Ad-hoc network.

2.2.1.2 Wireless networks

The two main properties of wireless networks [32] that differ from wired networks are the possibility of broadcast transmission and also interference. Wireless networks are constantly affected at diverse aspects, such as low throughput, dead transmition spots and inadequate ability for mobility.

NC brings to wireless networks a possibility of exploring the broadcasting of wireless nodes through the shared wireless medium providing benefits concerning end-to-end delay, bandwidth and transmission power effectiveness.

A developed research named the COPE project [33] demonstrated benefits of NC for wireless networks, at MAC layer, even using simple codification with XOR coding, and that is why [33] was named *"Xors in the air"*, making all the sense. Ahlswede et al.[1], on an early approach to wireless networks, ignored the interference upon a receiver. An example set at Physical Layer NC, represented in Figures 2.8 and 2.9, demonstrates the NC benefits compared to traditional routing.

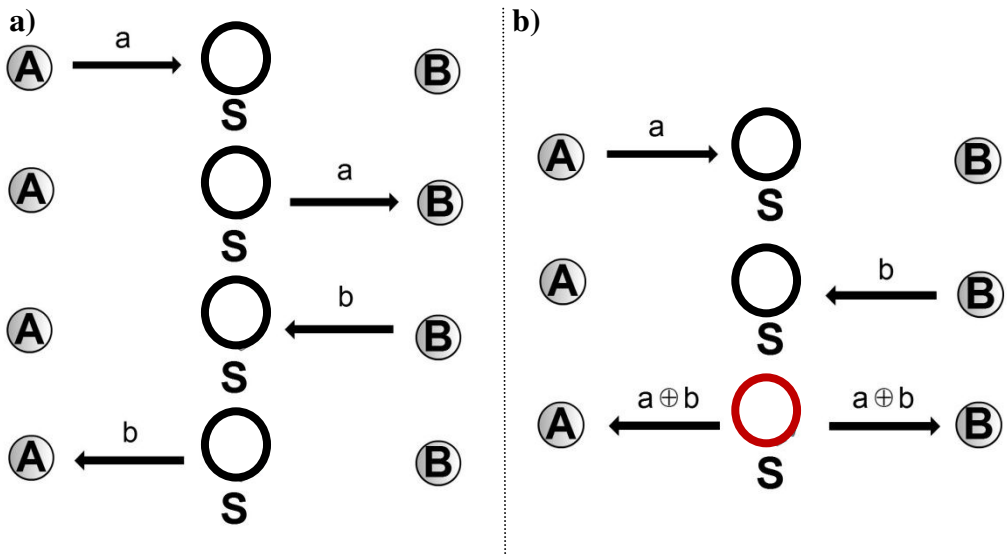

Figure 2.8 - a) Wireless network with traditional routing. b) Wireless network with NC broadcast.

In the example described in Figure 2.8, a typical NC wireless approach is set. There are three wireless nodes *A, B and S* where nodes A and B want to exchange packets via *S.* A traditional *store and forward* approach is described in Figure 2.8 a),

where the number of transmissions, for both nodes to receive both packets *a* and *b,* is four. In Figure 2.8 b), a wireless NC approach is described, where both packets are *XOR*ed in node S and consequently sent to both its outgoing edges, being three the number of transmissions.

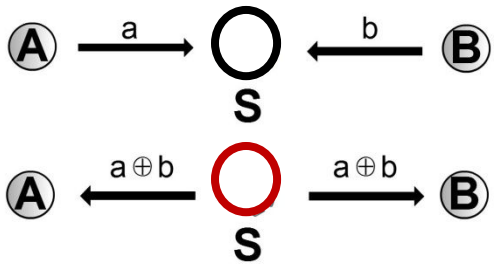

Figure 2.9 - Wireless Physical layer NC.

Two wireless nodes *X* and *Y* want to exchange packets via an intermediate node *S*. With physical layer NC and utilization of broadcast, the number of transmissions is two. Because of physical layer NC, the classical view of interference that it is harmful is changed,

2.2.2 Security (Secure Network Coding)

The example model set for exploring NC benefits within security in a network [8] is the same as network multicast using NC but, in terms of secure NC, the packets transmitted through the sets of edges can be wiretapped, deleted, observed by unauthorized users, or even altered by channel errors. Network codes can be designed for networks such as shown in Figure 2.10.

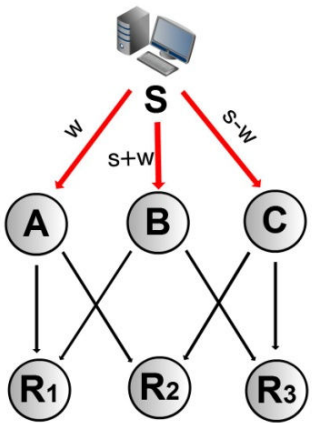

Figure 2.10 - Secure NC. Secure message *s* sent with randomness, *w*.

The network code has to be set in a way such that the multicast packet is information theoretically secure, independently of which compromised set of edges. As shown is Figure 2.10, the multicast information generated at the source *S,* (*s,w*), represent the secure packet *s* and the randomness *w*. Each edge in red can be compromised (wiretrapped), but since only one set can be accessed by the unauthorized user, with NC it can not be obtained any useful information about packet *s*.

Finally, it is easy to retrieve the secure information sent by *S* at any receiver node performing simple algebra calculations.

3 Simulation Models

Since they first appeared, Network simulators have aroused and improved their quality, performance, management and prediction tools over the past years.

The only few known NC simulations implemented are NECO (NEtwork COding) [19] and an OPNET (Optimized Network Evaluation Tool) NC tutorial by C. Lydick [7]. The last reference was the starting point in this thesis as there is no standard NC library for this platform or any other, being OPNET [27] the network simulator chosen to simulate the networks in this thesis.

In this chapter, a brief review of OPNET methodology, applied also with Matlab, is described.

3.1 *Why OPNET?*

Besides OPNET being one of the top most famous network simulator tool used both by commercial and research communities, having the possibility of having [7] as a starting point was a main reason as also as the opportunity to learn, study and explore the capabilities that this powerful tool offers. In Figure 3.1, a simple OPNET diagram, resuming an OPNET's project simulation process, is illustrated.

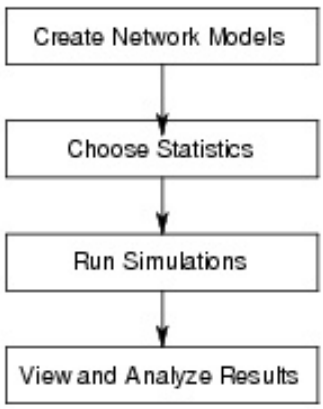

Figure 3.1 - OPNET's simple diagram [29].

3.2 *OPNET Network modeling overview*

To develop OPNET models and simulation scenarios, an object-oriented approach is used. Like in any other object-oriented programming language, the models, as a class, can be reused any time during the simulation.

The available basic models can be reprogrammed to satisfy the desired functions for the specific simulation, hence, the attributes of the customized models can be defined by the user in order to satisfy any kind of particular or enterprise specifications.

With OPNET, the steps used to build a network model and run simulations, workflow, take place around the Project Editor. Simulation models are organized, by hierarchal order, in two more main levels: node editor and process model editor.

3.2.1 The Project Editor

The Project Editor is the main stage for creating the desired network simulation scenario. At this editor, a network model can be built using the models available from the object palette, choose individual statistics about the network, run a simulation and view the results.

3.2.2 The Node Model

The Node Editor, at the second level in the hierarchy, defines the behavior of each network object, describing the diverse functions of the node.

Its function is defined using different modules, which are implemented using process models (lowest level in the hierarchy) and define the behavior of the node, as data creation, data process, data storage, etc.

The final network object function is made up by the multiple modules connected through packet streams and/or statistic wires, as illustrated in Figure 3.2.

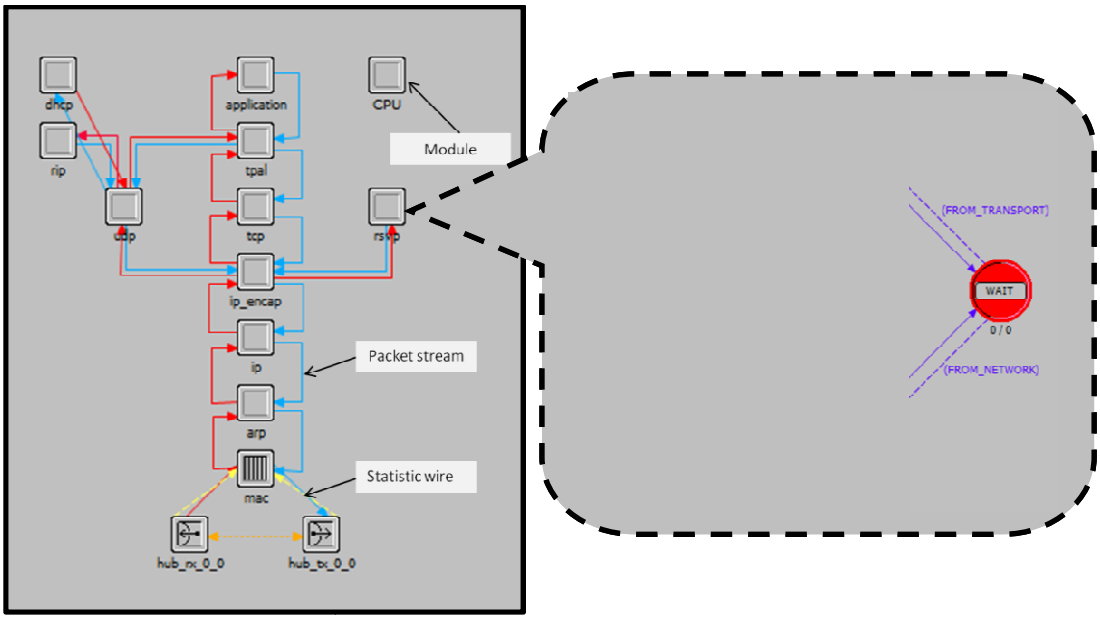

Figure 3.2 - OPNET's node model and process model snapshot, respectively.

3.2.3 Process Model

The Process model controls the defined functionality of the node models created. It is represented by finite state machines (FSMs), symbolized by icons that represent states and lines/links which represent transitions between states (Figure 3.2).

Each state or transition is programmed in C or C++ languages so, operations in each state or transition are described in C/C++ code blocks.

The defined structure of the models, together with the support for C language programming, allows an easy development of communication or networking models.

3.3 *OPNET/MATLAB interface*

In this thesis, an interface between OPNET and Matlab was studied in order to understand how to implement it. OPNET is known as a powerful simulation engine. Matlab is a known software used for many type of calculations, providing diverse types of toolboxes.

The interest of integrating Matlab with OPNET in this thesis, relies in calculating the desired Finite Fields in order to generate the global/local coding vectors in the created network.

In order to use Matlab functions within OPNET, the MX-interface (API)provided by MATLAB allows C programs to call functions, in OPNET theprocess models, developed in MATLAB. To get this interface working is necessary to follow some guidelines:

- In OPNET, including the following *lib* files in the *bind_shobj_libs* environment attribute: libmat.lib; libeng.lib; libmex.lib; libmx.lib. Include also their directory/path into the repository flags. A snapshot of the environment attribute's window is displayed in Figure 3.3.

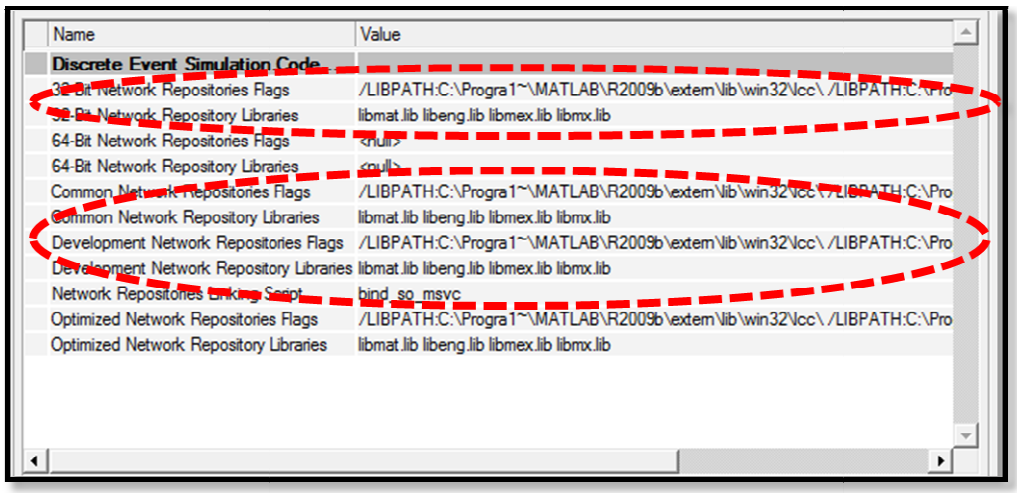

Figure 3.3 - OPNET's environment attribute snapshot including the essencial *lib* files and respective path.

- In a Windows OS, inserting MATLAB's directories, that contains the above files and the header file *engine.h*, in the environment variable *PATH*.

After these steps, the MATLAB engine can be called by OPNET with the function *engOpen,* which provides a pointer to a memory allocation that consequently forwards it to MATLAB engine.

Diverse functions can be called within MATLAB engine that allow a simple manipulation of variables or strings between platforms.

Some of the most common and studied functions in this thesis are displayed in the following Table 3.1.

MATLAB Engine function	Purpose
engClose	Quit MATLAB engine session
engEvalString	Evaluate expression in string
engGetVariable	Copy variable from MATLAB engine workspace
engGetVisible	Determine visibility of MATLAB engine session
Engine	Type for a MATLAB engine
engOpen	Start MATLAB engine session
engOpenSingleUse	Start MATLAB engine session for single, nonshared use
engOutputBuffer	Specify buffer for MATLAB output
engPutVariable	Put variables into MATLAB engine workspace
engSetVisible	Show or hide MATLAB engine session

Table 3.1 - MATLAB Engine functions

3.4 *Galois Field calculations with MATLAB*

As mentioned before, the OPNET/MATLAB interface in this thesis was taken in interest because MATLAB has the capacity of generating Galois Fields (Finite Fields) and being also able to perform calculations within them.

3.4.1 Galois Field functions

The main function used is ***gf(x,m)***, which creates a GF array with 2^m elements, with *m* an integer between 1 and 16, from the matrix *x*. The resulting output is a GF array which MATLAB recognizes and distinguishes from an array of integers.

At Table 3.2, is listed the primitive polynomial that the *gf* function uses by default, for each GF(2^m) created, is listed. With this table a better understanding of the defined polynomials within GFs, using MATLAB, can be achieved.

m	Default Primitive Polynomial	Integer Representation
1	$D + 1$	3
2	$D^2 + D + 1$	7
3	$D^3 + D + 1$	11
4	$D^4 + D + 1$	19
5	$D^5 + D^2 + 1$	37
6	$D^6 + D + 1$	67
7	$D^7 + D^3 + 1$	137
8	$D^8 + D^4 + D^3 + D^2 + 1$	285
9	$D^9 + D^4 + 1$	529
10	$D^{10} + D^3 + 1$	1033
11	$D^{11} + D^2 + 1$	2053
12	$D^{12} + D^6 + D^4 + D + 1$	4179
13	$D^{13} + D^4 + D^3 + D + 1$	8219
14	$D^{14} + D^{10} + D^6 + D + 1$	17475
15	$D^{15} + D + 1$	32771
16	$D^{16} + D^{12} + D^3 + D + 1$	69643

Table 3.2 - MATLAB *gf* function default primitive polynomial.

When using function ***gf*** to perform calculations between arrays, simple operators as "+", "-" or "*" can be used. As an example, the code below creates a GF(2^2) array, m=2, and adds another GF array with the same field, GF(4) :

```
x = 0:3;                    % [0 1 2 3] row vector
m = 2;                      % Work in the field GF(2^2) -> GF(4).
a = gf(x,m)                 % Creates GF array in GF(2^m)
b = gf([0,1,3,1],m)         %

c= a + b                    % Adds a to b, creating c.
```

```
%%%%% The resulting output is %%%%%

a = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)

Array elements =

   0   1   2   3


b = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)

Array elements =

   0   1   3   1

c = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)

Array elements =

   0   0   1   2
```

Making a simple analysis, the output indicates the values of GF arrays *a*, *b* and *c*, showing each output:

- The field, GF(2^2) = GF(4).
- The primitive polynomial for the field, by default for GF(4).
- The array of GF values each variable contains.
- The command that creates c shows how, having defined the variable b as a Galois array, you can add *a* to *b* by using the ordinary + operator. MATLAB performs the addition operation in the field GF(4). The output shows that *c* compared to *a*, is in the same field and uses the same primitive polynomial.
- For example, the second element of *c* is zero because the sum of any value with itself, in a Galois field of characteristic two, is zero, differing from the typical sum within infinite fields for integers.

To illustrate what the array elements in a Galois array mean, Table 3.3 lists the elements of the field GF(8) as integers and as polynomials in a primitive element, A. The table helps to interpret a Galois array like

```
gf8 = gf([0:7],3); % Galois vector in GF(2^3)
```

Integer Representation	Binary Representation	Element of *gf(8)*
0	000	0
1	001	1
2	010	A
3	011	A+1
4	100	A^2
5	101	A^2+1
6	110	A^2+A
7	111	A^2+A+1

Table 3.3 - MATLAB *gf(8)* elements.

Other functions for performing operations within GF can be considered using GF(p) arrays, where *p* stands for a prime number. In Table A.1 a brief resume of those functions description is made.

4 Implementation and results

All the simulations and implementations in this thesis were performed using the network simulator OPNET Modeler version 14.5 [9] and the developer of technical computing software, MATLAB version 7.9.0 [32].

4.1 *OPNET/MATLAB NC Tutorial*

This tutorial presented by UWEE FunLab [6], was studied and taken like a first step and an example for the simulations and implementations realized in this thesis, concerning OPNET and OPNET/MATLAB interface. The source code is available in [7]. In Figure 4.1, a snapshot of this tutorial Project Editor is displayed.

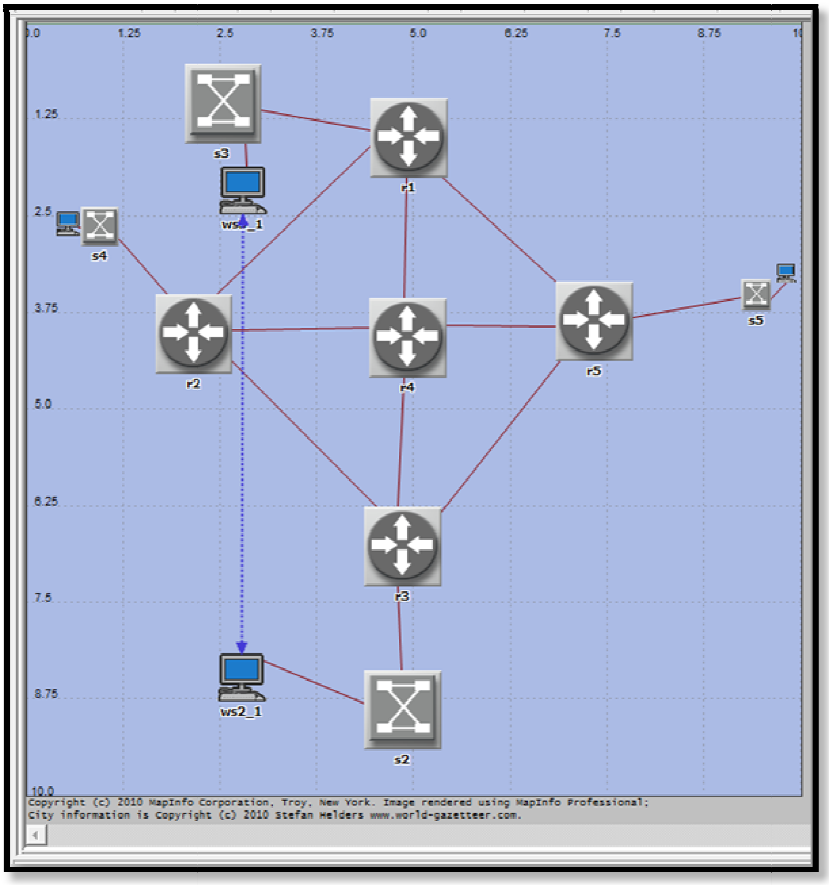

Figure 4.1 – OPNET Project window from OPNET NC tutorial, UWEE FunLab [7].

This project consists in implementing NC in a specific network topology, as defined in Figure 4.1, through OPNET, with the objective of investigating NC throughput.

As told in the past chapters, the basic idea of LNC is that intermediate nodes receive packets, process those packets and forwards the resulting packet through its outputs. In this tutorial, routers (nodes) receive packets, multiply them with chosen coefficients and forward the resulting packet. For each output link there is a unique NC coefficient.

The Ethernet station is the one generating the packets, so the first router does a NC operation and sends out three different packets on its three outgoing links. The same process occurs at other routers, as packets are identified as UDP packets from the NC tracing demand.

Each node model (router *"nc_router"*) in OPNET was changed in order to handle NC or non-NC packets in the network. To define the role of each processor (process model) role in the node model, interfaces which receive NC packets are defined as *RECEIVE* interfaces and those who send NC packets are defines as *SEND* interfaces, as non-NC packets (normal packets) are processed regularly.

By definition, NC is intended to implement over Network Layer (layer 3), through IP frames, however, in this project NC is employed over Transport Layer (layer 4) since it is easier to implement as it allows adding a kind of hidden data within the TCP/UDP frame in OPNET, being invisible to the user and not entering in the simulation statistics as the packet format includes a *flag bit* within the UDP header whether the packet is a NC or non-NC packet.

A *m-bit* field ($\mathbb{F}_{2^m}$) is assigned within the TCP/UDP header for the LNC, called exactly *LNC field*, as illustrated in Figure 4.2.

Figure 4.2 - NC Packet format [6].

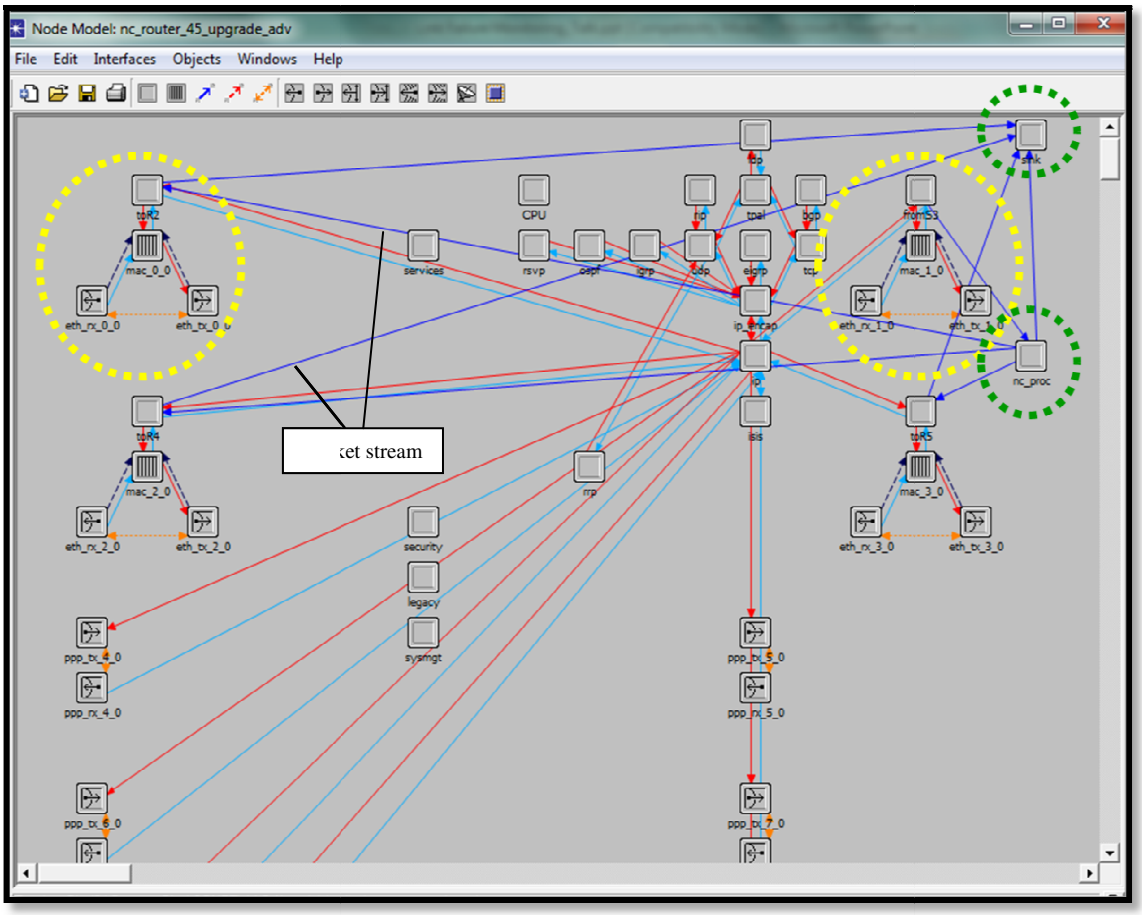

Figure 4.3 - OPNET NC Node Model: in yellow, *SEND/RECEIVE* interfaces, in green, *sink* and *nc_proc* processors.

In Figure 4.3, a snapshot of the Node model of the projectis displayed, dark blue arrows represent packet streams identified as UDP packets. As explained before, NC operates on unidirectional links so each interface is set as *SEND* or *RECEIVE* interface, where *SEND* interfaces will assume NC packets in error and consequently forward to the *sink* processor. *RECEIVE* interfaces forward NC packets to the *nc_proc* processor which basically copies the packet and sends it to its output streams with OPNET function,

*op_pk_send(op_pk_copy(*Packet),output_streams);*

The processor *nc_proc* is implemented dependently of the topology of the network created as it requires having a constant number of outputs so it can remain constant to all routers, therefore the necessity to send some of the packets to the *sink* processor.

The Node model role could be described, in a very simple approach, as illustrated in Figure 4.4.

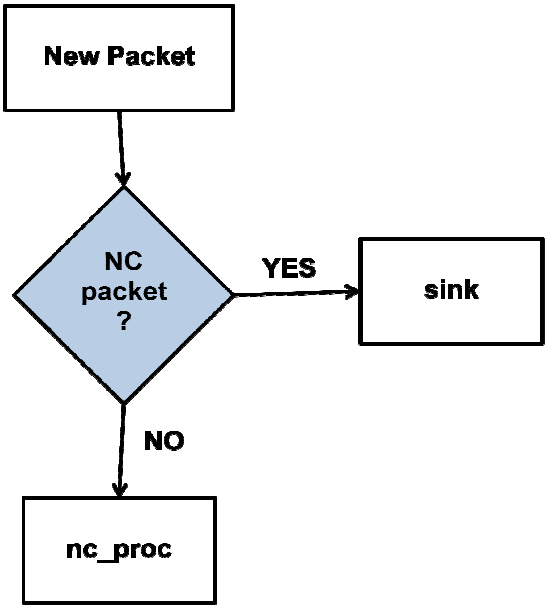

Figure 4.4 - Node model diagram.

At the processors, functions are implemented in order to obtain the NC packets, place them in queues so that later they can be combined and sent along the *SEND* interfaces. This is where timeouts in the network take place, in some routers this timeout is unnecessary as NC packets sometimes only need to be copied, combined with *nc_value,* defined in the *SEND,* interface and sent to the next router.

Other parameters can be defined in the *SEND* interfaces, *Bitsize of NC Value* and *Output NC value.* As the names indicate, the size of the code used in the Finite Field operations can be defined in the *Bitsize of NC Value* as the *Output NC value* defines how the *nc_value* is calculated, setting its value randomly as a 0 or specifying the value as desired.

The defined timeouts and packet flows for this network can be better understood in Figure 4.5.

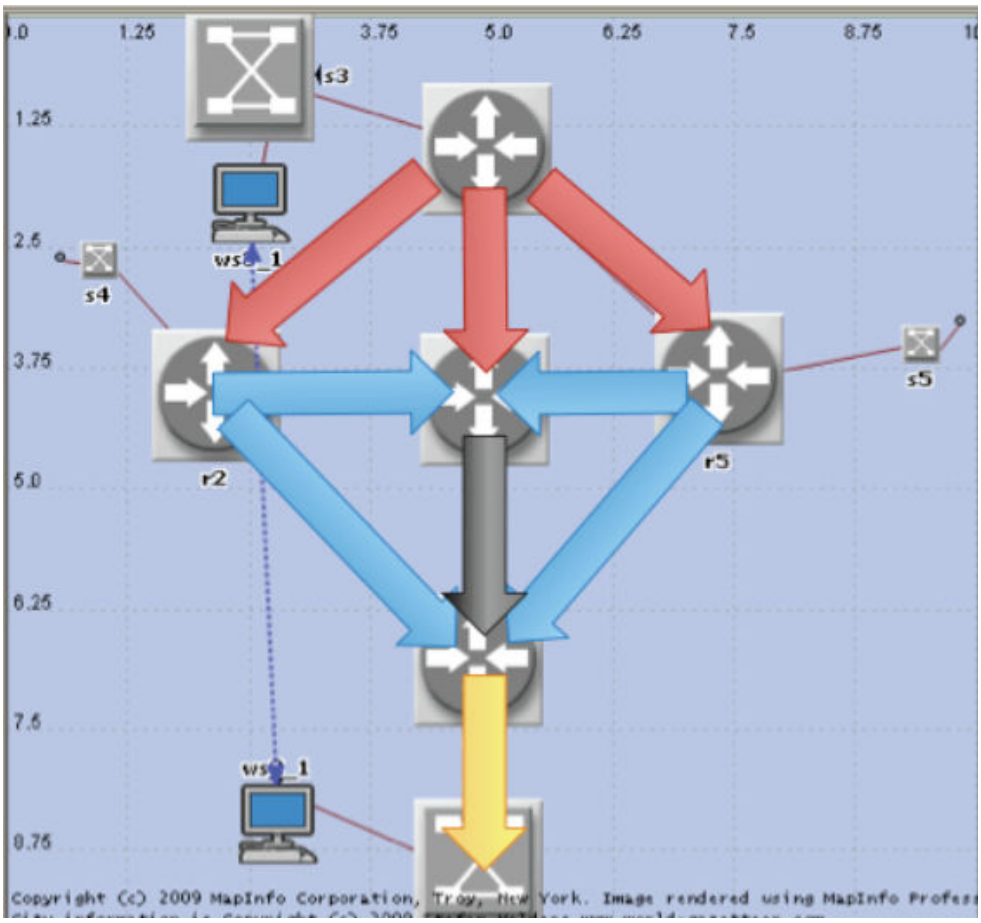

Figure 4.5 - Timing and Project's packet flows [7].

From a quick analysis of Figure 4.5, the packet flow within the network can be easily understood and consequently, also its timeouts. Red arrows represent the packets flowing from R1. Since it only receives them from a single interface, it does not need to wait to the send the packets. The same happens to routers R2 and R5 as they only receive packets from one interface.

Router R4 is different as it will receive firstly the packet from the red flow (R1) and later the ones from the blue flows (R2 and R5), therefore a timeout must be defined so that packets can be enqueued, combined and sent to R3. Same case for router R3.

As C. Lydick says in [7], a simple way to think is to divide each packet flow in time steps, red arrows occur at time step one, blue at time step two, black at time step three, and yellow at time step four.

A way to simulate how the network would respond to link failure is simply to define the timeout, in the pretended interface, with an infinite value, such as 100.

4.1.1 Galois Field operations

After each timeout, the interface dequeues all packets received and combines them using finite fields operations with MATLAB function *gf()*, through its mx-interface.

```
y=0;
y = gf(y, nc_bitsize) + gf(packet_received, nc_bitsize);
```

Performing the above calculations *x* times depending on the how many packets each interface received. For example for R1, *x*=1 and for R4, *x*=3. Afterwards, the resulting value is multiplied with the output *nc_value* defined.

```
y = gf(y, nc_bitsize) * gf(nc_value, nc_bitsize);
```

4.1.2 Results

After each simulation being performed, or during it, the console output can be verified, taking the form:

(Current simulation time | Process ID) At <NC Value at the output interface> = Value of packet after combination and multiplication of nc value at output.

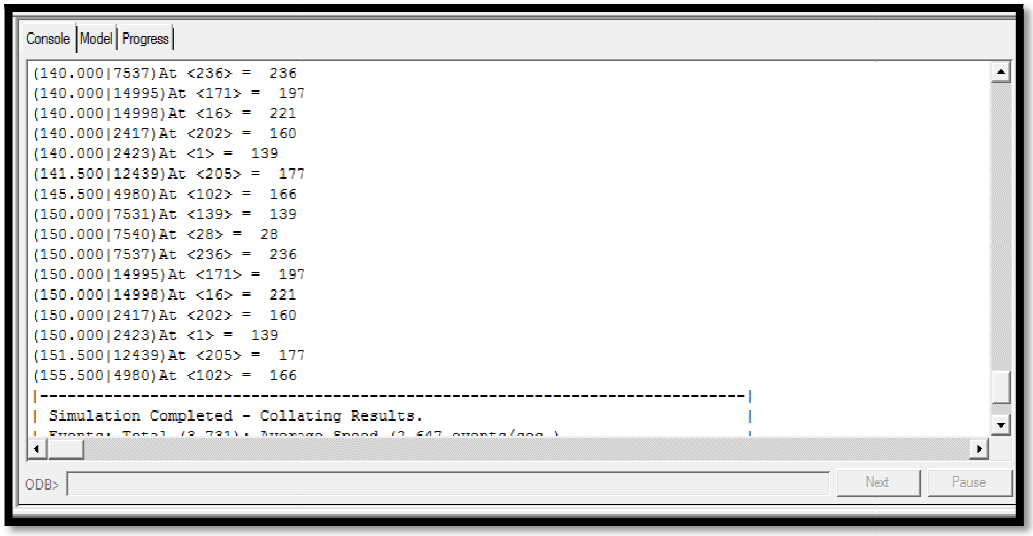

Figure 4.6 - Simulation Console: Final NC packet values.

The NC values used in [7] were set and the calculations were performed first through MATLAB and then through the simulation within OPNET to compare with the resulting values obtained.

In Figure 4.6, a snapshot of the simulation console of the project is displayed. With the notation set above, it can be verified for example that at time 140 (seconds), the value at processor ID 14995, which refers to node 5 in Figure 4.7, with output nc-value of 171, gives a nc-value product of 197.

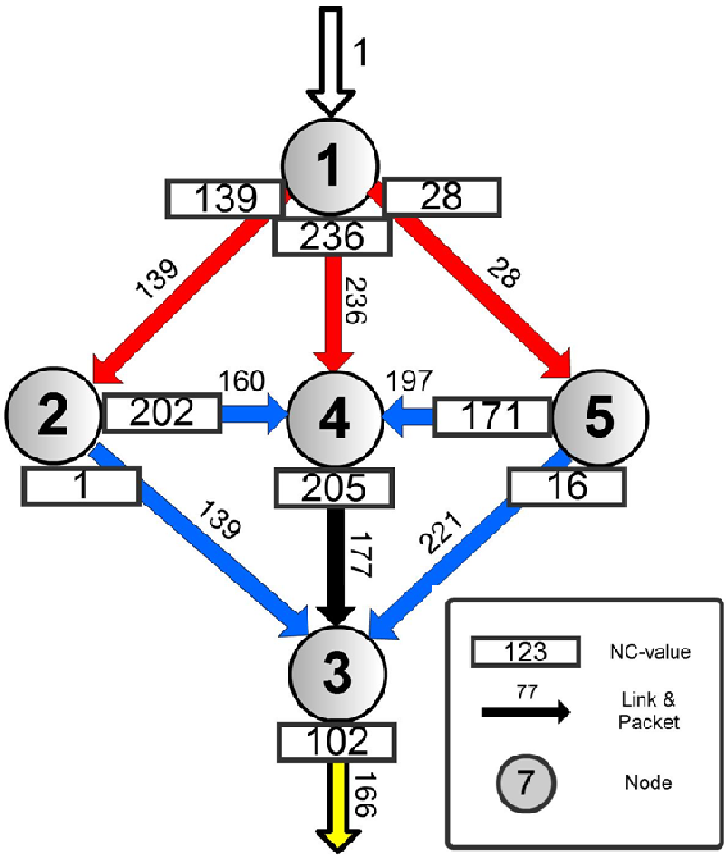

Figure 4.7 - NC values and Packet Values for OPNET simulation. Adapted from [7].

The individual statistics chosen to express the throughput within links/edges is displayed through the graphic in Figure 4.8 as the packet flow in the network is shown in Figure A.1. The resulting throughput curves are not conclusive for proving a throughput enhancement in the network as NC operations are applied in every node.

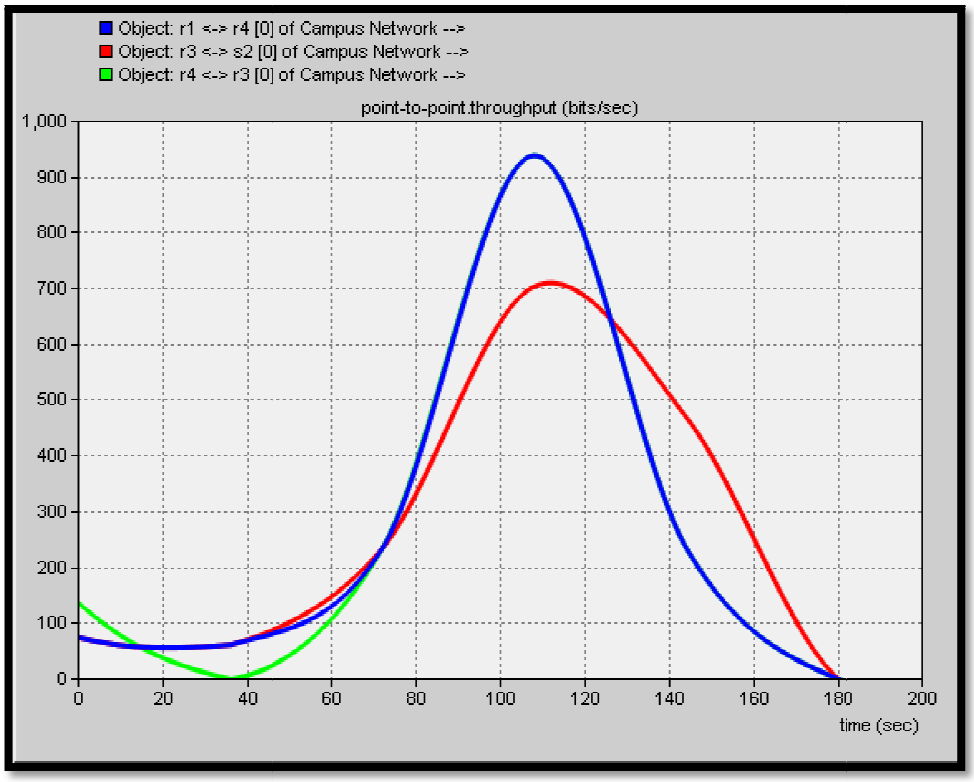

Figure 4.8 - Throughput statistics through selected links.

4.2 *Linear Information Flow*

Considering an acyclic graph $G= (V, E)$, the LIF algorithm [15] is called a trendy algorithm as it basically consists in finding *global coding vectors* for each edge $e \in E$, in G. It is a polynomial time algorithm drawn toward multicast code construction which was chosen to explore, study, implement (MATLAB) and simulate (OPNET(C/C++)) in this thesis.

4.2.1 The algorithm

Every coding in the algorithm [15] is linear with symbols from $\mathbb{F}$, with size $|\mathbb{F}|=2^m$ as it performs linear combinations of every packet or information received in any node, LNC.

The algorithm consists in two main stages. The first is expressed by finding the max-flow h for each sink/receiver $t \in T$, defining a set of h edge-disjoint paths, f^t, from

source S to each $t \in T$. The edges belonging to each edge-disjoint path defined are considered in the second stage.

Going through the nodes in topological order[1] allows that, when in each step it is codified the outgoing edges of the respective vertex/node $v \in V$, it is known that all the incoming edges of that node have already been codified before. The codification selected to an outgoing edge is based in the theory that if the information stream selected to the edge from the source *s* to the receiver *t, min-cut,* are linearly independent, then the receiver can decode it and reconstruct the information sent by the source.

To describe the algorithm in a more specific approach, considering receiver $t \in T$, let $C_t \subset E$ be any set of *h* edges such that it contains one edge from each *h* edge-disjoint paths from *s* to *t,* corresponding *min-cut,* and B_t an *hxh* matrix. Hence, if the information sent through the edges $c \subset C_t$ are linearly independent, receiver *t* can reconstruct the message originally sent by source *s*.

If an edge *c* belongs to a path for more than one receiver, the linear independency for that edge must be verified to each receiver.

For B_t, its *h* columns correspond to the *h* edges of C_t, and the column for edge $c \in C_t$ represents the linear combination of $b_1,\ldots,b_h$ that flows through edge *c*, where $b_1,\ldots,b_h$ correspond to the original source symbols. As edge *c* transports $b_c(1)b_1+\ldots+b_c(h)b_h$, the corresponding column for that $c \in C_t$ will be $[b_c(1)+\ldots+b_c(h)]^T$.

Thus, it is ensured that B_t is, at every time step, invertible, permitting that the original source symbols $b_1,\ldots,b_h$ arriving at each sink $t \in T$, remain recoverable with every coding vector formed.

To better explore the algorithm, following the notation in [15], let $\Gamma_I(v)$ represent the set of incoming edges in node *v*; $\Gamma_O(v)$ represent the set of outgoing edges in node *v* and *start(e)* represent the node where edge *e* starts. Over each edge *e* it is set the length *local coding vector* - $|\Gamma_I(start(e))|$, where

$$m_e : \Gamma_I(start(e)) \longrightarrow \mathbb{F}^{\Gamma_I(start(e))},$$

as the vector that allows to determine the linear combination of the symbols on the edges of $\Gamma_I(start(e))$, generating the symbol on edge *e*. Considering that y(e) is the symbol transported by edge *e,* it is given

[1] A topological order exists for every edge of an acyclic graph *G*. It is simply a partial order.

$$y(e) = \sum_{p \in \Gamma_I(start(e))} m_e(p)\, y(p). \tag{4.1}$$

The objective here is to determine the coefficients $m_e(p)$ in order that all sinks can successfully recover the original symbols from the incoming packets.

4.2.2 Network topology

The network topology studied in order to simulate and implement the LIF algorithm, was chosen from [3] with the intention of comparing the results obtained for this network using the LIF algorithm and the given approach from [3].

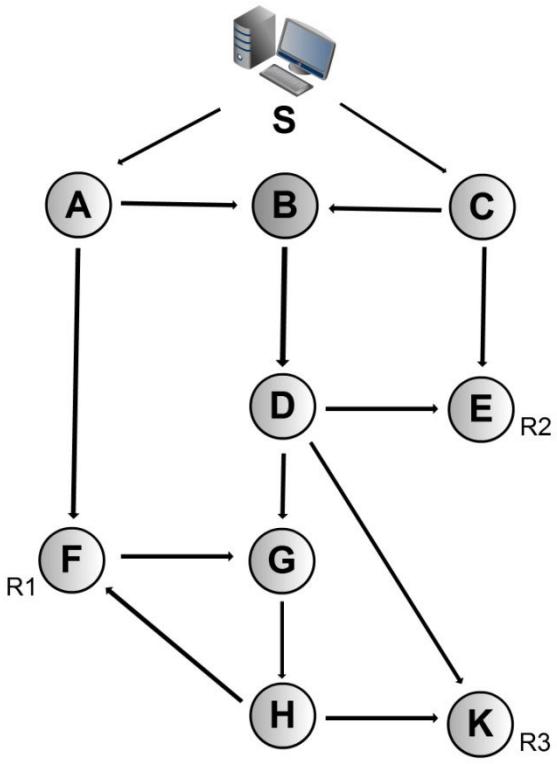

Figure 4.9 - Network topology.

The chosen network topology is represented in Figure 4.9. To analyze the network graphic $G=(V,E)$, firstly it must be found h edge-disjoint paths connecting each sink/receiver to the source. In Figure 4.10 are shown the two edge-disjoint paths for

each receiver, being also easy to notice that the paths to different receivers collapse over edges BD and GH.

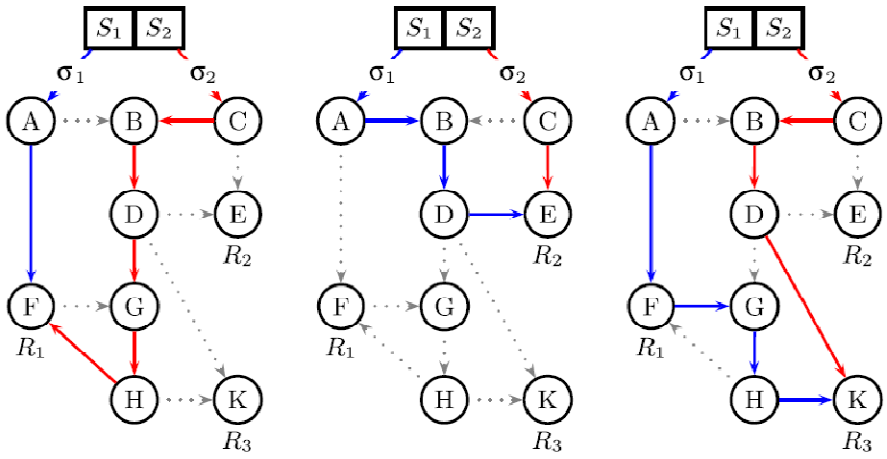

Figure 4.10 - Edge-disjoint paths to each receiver [3].

If the transmission was restricted to traditional routing, when symbols σ_1 and σ_2 would share the same unit capacity edge, it could only be possible to forward only one of the symbols. As said before in chapter 2, using LNC to combine both symbols over a finite field of size, $\mathbb{F}_q$, that transmission would become possible.

4.2.2.1 Related work using LNC

In [3] a LNC approach is made to the referred network described in Figures 4.9 and 4.10. It basically consists on assigning a *local coding vector* m_e for that edge which is being shared by the two symbols, that are multiplied to the incoming symbols of edge e, $\Gamma_I(e)$, performing the linear combination referred before. The *local coding vector* m_e, vector of coefficients over $\mathbb{F}_q$, has a 1 x $|\Gamma_I(e)|$ dimension.

Through the analysis of this network, edges BD and GH are *coding points*, the edges where symbols collapse and therefore need to linearly combine the received symbols. In [3] approach, since the value of the local coding vectors are unknown, they are assigned as

$$m_{BD} = [\alpha_1 \ \alpha_2] \ ; \ m_{GH} = [\alpha_3 \ \alpha_4]$$

This LNC solution is described in the following Figure 4.11. In this solution the vector of coefficients of the source symbols linearly combined with the respective *local coding vector* m_e, assigned to an edge *e,* are defined as *global coding vector* b(*e*). The dimension of b(*e*) is 1 x *h*, so it would be a *h*-dimensional vector over $\mathbb{F}_q$.

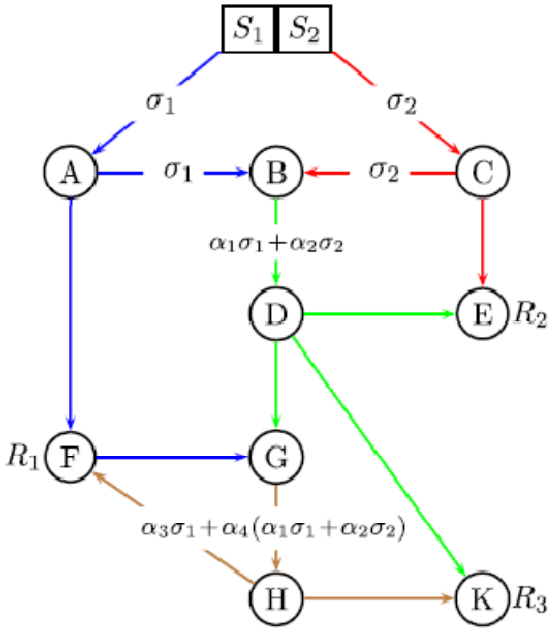

Figure 4.11 - Linear Network Coding (LNC) solution. Coding points: edges BD and GH [3].

The resulting global coding vectors joined together with the input edges of a receiver, defines a system of linear equations which the receiver can solve with a simple *Gaussian* elimination and retrieve the original source symbols.

4.2.3 Implementation

As said before, the implementations of the algorithms were made with MATLAB, mainly because of the ability of performing operations within finite fields with a simple function call.

The LIF algorithm [15] is summarized in the following Algorithm 4.1. It was the most studied algorithm, being the implementation of it now described. As said before, the chosen network topology to implement the LIF algorithm is described in Figures 4.9-4.11.

ALGORITHM 4.1: LINEAR INFORMATION FLOW

1: $h \Leftarrow \min_{t \in T} |\text{max flow from } s \text{ to } t|$
2: insert a new source s' into V
3: insert h edges $\{ e_1,\ldots,e_h \}$ from s' to s into E
4: $f^t \Leftarrow$ set of h edge-disjoint paths from s to t // chosen flow from s to t
5: $\mathbb{F} \Leftarrow$ finite field size $\geq 2\,|T|$
6: **For all** i **do:**
7: $\quad b(e_i) \Leftarrow [0^{i-1}, 1, 0^{h-1}]$ // i-th unit vector of $\mathbb{F}^h$
8: **For all** *sinks* ($t \in T$) **do:**
9: $\quad C_t \Leftarrow \{e_{1,\ldots,} e_h\}$ // t is supplied through C_t
10: $\quad B_t \Leftarrow \{b(e_1),\ldots, b(e_h)\}$ // coding vectors
11: $\quad$ **For all** *vectors* ($c \in C_t$) **do:**
12: $\quad\quad a_t(c) \Leftarrow b(c)$ // inverse vectors
13: **For each** *vertex* ($v \in V \setminus \{s'\}$) **do:** // in topological order
14: $\quad$ **For all** $\Gamma_O(v)$ **do:**
15: $\quad b(e) \Leftarrow \sum_{p \in P(e)} m_e(p)\, b(p)$ such that //choose linear combinations
16: $\quad\quad$ **For any** $t \in T(e)$
17: $\quad\quad B_t \setminus \{b(f^t_{\leftarrow}(e))\} \cup \{b(e)\} \Leftarrow$ is linearly independent
18: $\quad$ **For all** *sinks* ($t \in T(e)$) **do:**
19: $\quad C'_t \Leftarrow C_t \setminus \{f^t_{\leftarrow}(e)\} \cup \{e\}$ //advance set of edges C_t
20: $\quad B'_t \Leftarrow B_t \setminus \{b(f^t_{\leftarrow}(e))\} \cup \{b(e)\}$ //update B_t
21: $\quad a'_t(e) \Leftarrow (b(e).\, a_t(f^t_{\leftarrow}(e)))^{-1}\, a_t(f^t_{\leftarrow}(e))$ //update a_t
22: $\quad\quad$ **For all** $c \in C_t \setminus \{f^t_{\leftarrow}(e)\}$:
23: $\quad\quad a'_t(c) \Leftarrow a_t(c) - (b(e).\, a_t(c))\, a'_t(e)$
24: $\quad\quad (C_t, B_t, a_t) \Leftarrow (C'_t, B'_t, a'_t)$

For a better approach and understanding of the algorithm, it was divided in six stages:

i. Find max-flow (line 1, Algorithm 4.1)
ii. Add artificial edges (lines 2-3, Algorithm 4.1)
iii. Define source → receiver path (lines 4-5, Algorithm 4.1)
iv. Set initial vectors that span $\mathbb{F}^h$ (lines 6-12, Algorithm 4.1)

v. Find resulting *global coding vector* b*(e)* for all receivers, linearly independency test (lines 13-17, Algorithm 4.1)

vi. Update variables (lines 18-24, Algorithm 4.1)

Firstly, it was considered the sources S_1 and S_2, from the original topology of the network described in Figure 4.11, as an unique source *s* multicasting the input symbols/packets through its two outgoing edges, as shown in Figure 4.9. Therefore, having an acyclic graph *G= (V, E)* as described in Figure 4.9, dealing with the underlying cycle {*FG, GH, HF*} with the introduction of memory for node F for example, it is determined the max-flow *h* for each receiver $t \in T$, defining a set of *h* edge-disjoint paths, f^t, from source *s* to each $t \in T$. From Figure 4.10 it was shown that there are two edge-disjoint paths for each receiver *t* and hence it is set *h*=2. This being set, the algorithm outputs a linear network code that guaranties the delivery of the two (*h*=2) symbols to all receivers $t \in T$.

An artificial source $s' \in V$ is created in the beginning. This allows inserting two edges $\{e_1, e_2\} \in E$ from *s'* to *s*, carrying the input symbols for the source *s*. To get access to the flows, the receivers supplied through edge *e* are set as *T(e)* and the predecessor edges of *e* in some flow path $\{f_{\leftarrow}^{t}(e): t \in T(e)\}$ are set as *P(e)*.

The finite field is defined satisfying *[15, Theorem 3],* which states that any finite field of size $|\mathbb{F}| \geq 2\ |T|$ can be used and hence sufficient for fulfilling the condition that B_t matrices are always invertible at each receiver *t*. As

$$T = \{F,E,K\} \implies |T|=3$$

a finite field of size $|\mathbb{F}| \geq 6$ was chosen and set as $\mathbb{F}_7=\{0,1,2,3,4,5,6\}$, completing the first three stages of the Algorithm 4.1.

Afterwards, it is defined the *t* initial B_t matrices 2x2 (*h* x *h*) set by the *global coding vectors* b(*e*) at the last *h* visited edges, in the flow f^t, which define the C_t matrix that supply the receiver *t*. At the beginning of the algorithm, each B_t matrix is defined by (4.2), an *h*-length vector with a 1 in the *i*-th location, corresponding to the vectors

$$b(e_i) = [0^{i-1}, 1, 0^{h-1}] \tag{4.2}$$

which span $\mathbb{F}^h$, therefore, as *h*=2, the initial B_t matrices would be defined as the identity matrix $I_2= \begin{bmatrix}10\\01\end{bmatrix}$, and each $C_t = \{e_1, e_2\}$. For all edges $c \in C_t$, a vector $a_t(c)$ is created for

assuring the linear dependency of the B_t matrix, being initially set at this stage as equally as the corresponding b(c) vectors.

In the fifht stage, the nonzero coefficients for the *local coding vectors* m_e are defined from $\mathbb{F}_7$ therefore, the calculation of the respective *global coding vectors* b(e) for edge e take place over $\mathbb{F}_7^2$ (A.1), defined by (4.3).

$$b(e) = \sum_{p \in \Gamma_1(start(e))} m_e(p) b(p), \qquad \text{for } e \in E \tag{4.3}$$

As said before, the created source *s'* at beginning of the algorithm supplies the original symbols through edges $\{e_1, e_2\}$ which are then processed as (4.3), sending through its output edges the resulting global coding vectors b(e) as explained individually for each node in Figure 4.12.

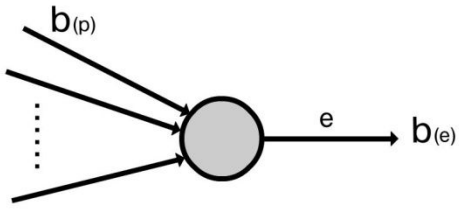

Figure 4.12 - Individual b(e) process at each node $v \in V$.

In the following Figure 4.13, it can be better understood the operations developed within the network.

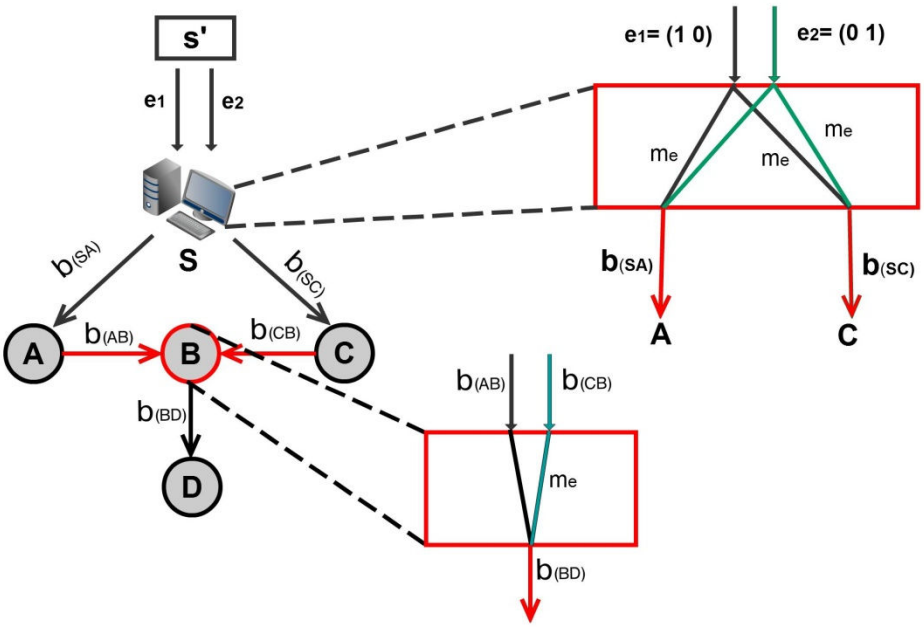

Figure 4.13 – Example scheme of the calculations performed in nodes S and B, with resulting b(e) vectors.

Finally, matrices B_t, C_t and a_t are updated along the algorithm, as it runs in a topological order. The matrix C_t exchange its column referring to the predecessor edge $f_{\leftarrow}^{t}(e)$ by the actual visited edge e, the final B_t matrix exchange its column referring to the global coding vector of the predecessor edge $b\left(f_{\leftarrow}^{t}(e)\right)$ by the b(e) calculated for the actual edge e, in order that the final B_t matrix is always linearly independent and a_t is defined by (4.4).

$$a'_t(e) = \frac{a_t\left(f_{\leftarrow}^{t}(e)\right)}{b(e).a_t\left(f_{\leftarrow}^{t}(e)\right)} \tag{4.4}$$

In the final, if the vectors b(e) are set as the rows of B_t and a matrix A_t is set with columns of a_t, then the invariant (4.5) is verified over $\mathbb{F}_7$.

$$A_t = B_t^{-1} \tag{4.5}$$

4.2.3.1 Results

As in MATLAB every variable has to be assigned, the implemented network topology was designed with *Matgraph* [40] to better understand in which node/edge the calculations were taking place, resulting the output graph of the network shown in Figure 4.13.

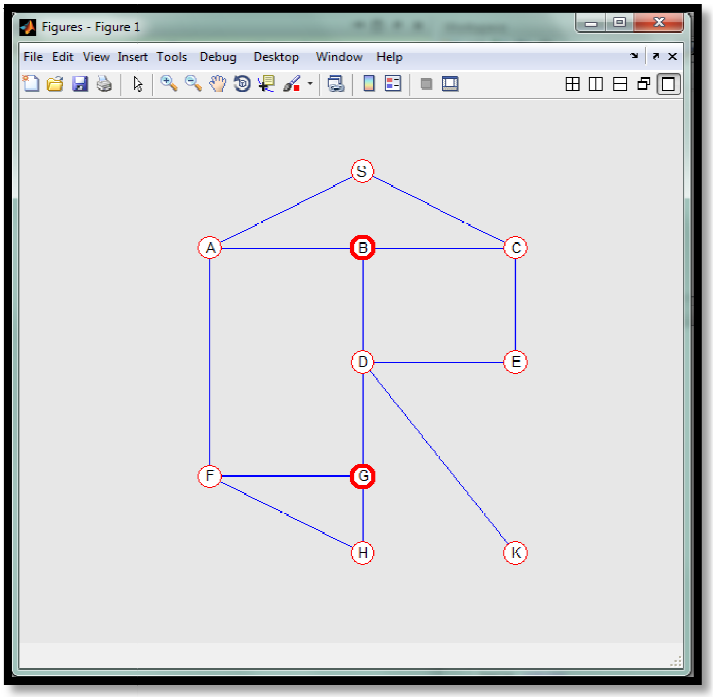

Figure 4.14 - Figure of the topology of the network within MATLAB.

Defining the variables in the beginning of the algorithm as, for example, T={F,E, K} ; Γ_I(E) ={*CE, DE*} ; Γ_I(K) ={*DK, HK*} ; start(*GH*)=*G ; T*(*DG*)={F,K} ; $f_{\leftarrow}^{K}(BD)$=b(*CB*).

When the algorithm reaches node B, the resulting *global coding vector* b(*BD*) will be defined as

$$b(BD)= \sum_{p\in\Gamma_I(B)} m_{BD}(p)b(p) = m_{BD}(AB)b(AB)+m_{BD}(CB)b(CB) \qquad (4.6)$$

As the algorithm finishes processing and hence reaches all receivers, having defined the *local coding vectors* m_e in the beginning, in topological order, as show in Figure 4.15:

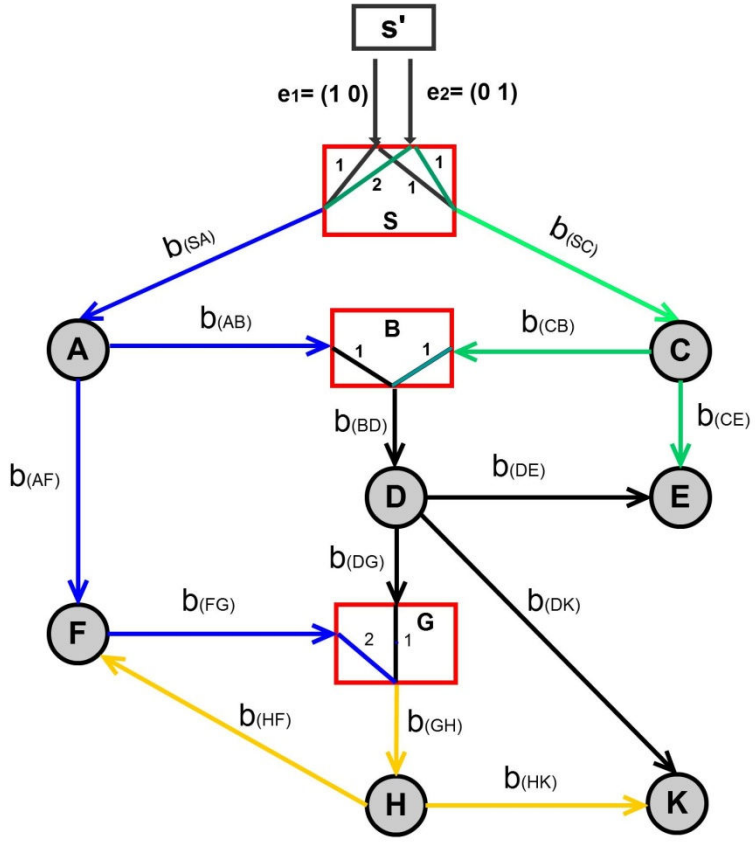

Figure 4.15 - Defined local conding vectors (me) in nodes S, B and G.

- [$m_{SA}(e_1)$=1, $m_{SA}(e_2)$=2]
- [$m_{SC}(e_1)$=1, $m_{SC}(e_2)$=1]
- $m_{AF}(SA)$=1; $m_{AB}(SA)$=1

- $m_{CB}(SA)=1$; $m_{CE}(SA)=1$
- $[m_{BD}(AB)=1, m_{BD}(CB)=2]$
- $m_{DG}(BD)=1$; $m_{DK}(BD)=1$; $m_{DE}(BD)=1$
- $m_{FG}(AF)=1$
- $[m_{GH}(DG)=1, m_{GH}(FG)=2]$
- $m_{HF}(GH)=1$; $m_{HK}(GH)=1$

After the calculations, always over $\mathbb{F}_7$, as an example, the global coding vector for edge *BD*, b(*BD*), will be

$$b(BD)= m_{BD}(AB)b(AB)+m_{BD}(CB)b(CB) = 1.[1\ 2] + 2.[1\ 1] = [3\ 4]$$

The final resulting matrices B_t and C_t for each receiver $t \in T$, after performed all calculations, over $\mathbb{F}_7$, are:

- For R_1,

$$B_{R\,1}= \left[\binom{1}{2}\binom{5}{1}\right] \text{ and } C_{R1} = \{AF, HF\}$$

- For R_2,

$$B_{R2} = \left[\binom{3}{4}\binom{1}{1}\right] \text{ and } C_{R2} = \{ DE, CE\}$$

- For R_3,

$$B_{R3} = \left[\binom{5}{1}\binom{3}{4}\right] \text{ and } C_{R3} = \{HK, DK\}$$

The original input symbols can then be easily retrieved solving the following system of linear equations:

$$\begin{bmatrix} y(c_1) \\ y(c_2) \end{bmatrix} = \boldsymbol{B_t} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4.7}$$

Where the *i*-th row of $\boldsymbol{B_t}$ is the *global coding vector* b(*c*) of the last visited edge $c \in C_t$, y(*c*) the last symbol carried on the last edge $c \in C_t$ and x_1 and x_2 the original input symbols transmitted.

Therefore, as shown before,

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

4.3 *Simulation methodology*

Within OPNET simulator it was implemented the network topology studied in this chapter and described in Figure 4.11.

The Node Model of the source node *S* is set by two process models which generate two different packets, defined as integers, multicasting through its output edges *SA* and *SC*. The Project model is shown in Figure 4.16a and the packet flow in the network, presented by the Animation Viewer application within OPNET is shown in Figure 4.16b.

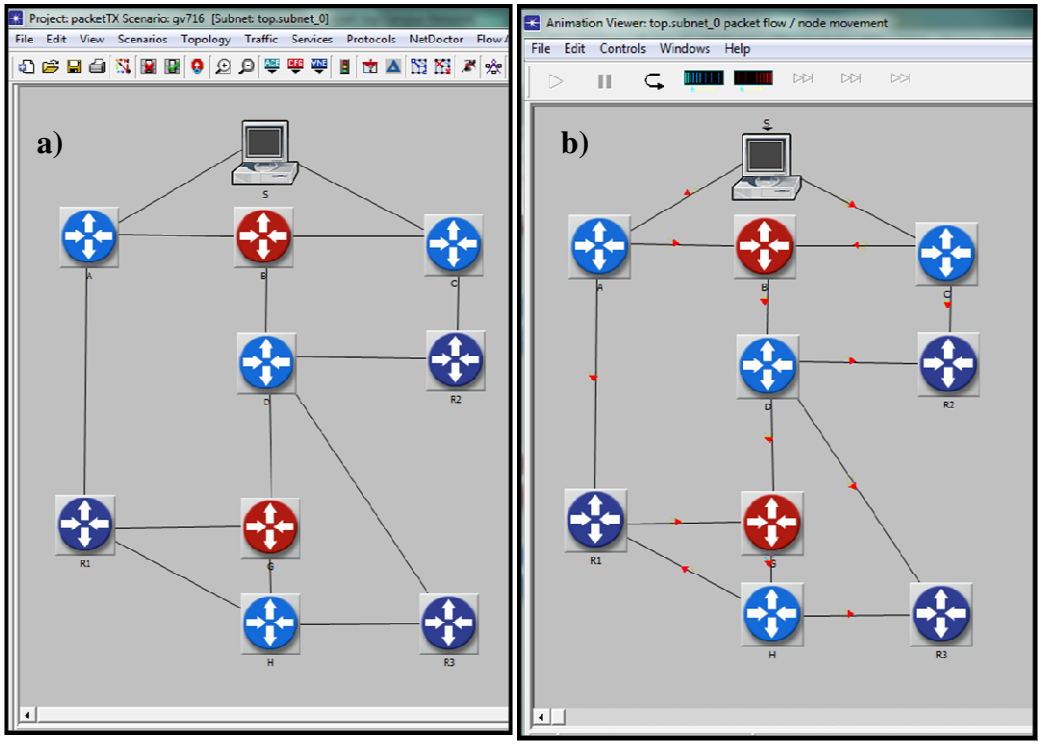

Figure 4.16 - a) OPNET Network Project editor. b) OPNET Animation Viewer. Packet flow in the network.

It was created as based model for each other Node Model a simple sink process model, with respective point-to-point transmitters and receivers, available in the palette of OPNET models, and then changed in order to execute a simple *store-and-forward* situation in such a way that packets could flow in the network as desired.

In nodes B and G, represented in red in Figure 4.16, these sinks were replaced by FIFO buffers, named "*NC_processor*", which store the two received packets through the incoming edges, performing LNC operations between them, as in Figure 4.11, and then forwarding the resulting packet through its output edges. Such Node Model and Process Model are shown in the following Figure 4.17.

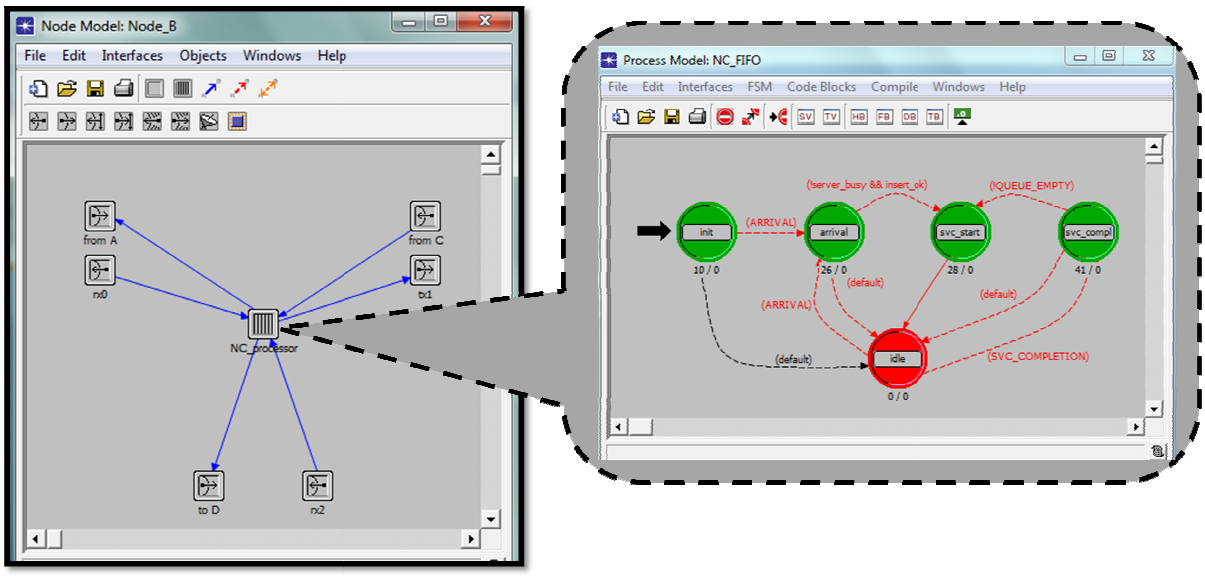

Figure 4.17 - Node Model for node B and respective NC_processor Process Model within OPNET.

In the end of the simulation, OPNET displays many statistical options. The throughput (bits/sec) in each edge can be selected but not for the entire network therefore only the size of the packets transmitted through each edge is displayed along the time of the simulation, as drawn in Figure 4.18.

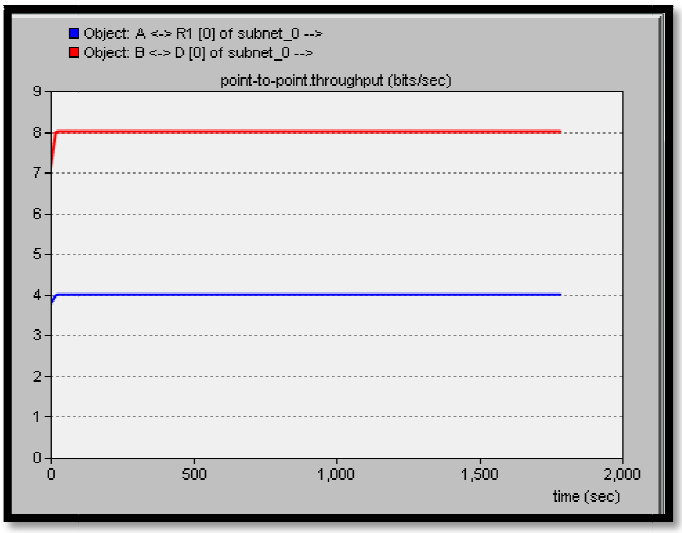

Figure 4.18 - Throughput in edges AR1 and BD.

5 Conclusions

5.1 *Conclusions*

This thesis has investigated *Network Coding* (NC) influence and characteristics. The objective of this thesis was to develop a simulation of a specific network in OPNET performing NC operations and implement NC algorithms such as the *Linear Information Flow* algorithm (LIF) in MATLAB.

Through the implementation of the LIF algorithm it was observed the effectiveness of NC within a network in such a way that the throughput of the network really increases as the intermediate nodes when faced with multiple incoming edges, combine the incoming packets allowing the transmission of only one combined packet avoiding the unnecessary need of sending each packet received in different time steps, forcing to a throughput decrease as the transmission time of the packets through the network increases. The robustness of the network and hence its security, is also proven in such a manner that the received packets at the receiver nodes can only retrieve the original sent packets solving a system of linear equations with the diverse coding vectors for each receiver calculated through each flow between source and receiver, as every calculations are performed using a specific finite field size.

As the simulation developed by C. Lydick [7] is, so far, the only developed example for the use of NC characteristics in OPNET, and as his tutorial was only designed for that specific network , the unsuccessful try of reaching the author lead that the use of MATLAB/OPNET interface used for NC operations developed in [7], was only useful to manipulate that specific network and understand the NC calculations performed at each node.

Through the OPNET simulation implemented it was observed how packets do flow in the chosen network topology and its consequences. As OPNET only states the

resulting throughput statistic for each desired edge, an overall network throughput improvement was unable to demonstrate, so the simulation would not bring any conclusive observation.

Network Coding is really a very interesting and enticing research area with many different kinds of applications being discovered, performed and experienced since it was first introduced. I have no doubts that this technique will represent more and more an important role in networking algorithms as it is already being approached by many important enterprises.

5.2 *Future Work*

With this thesis I was introduced to OPNET and found that there is indeed a steep learning curve to get to a point that you can confidently perform the needed implementations in the Process Models of OPNET in order to produce the desired simulation and results. This being said, a long time was taken to figure how to implement a NC algorithm in a specific network topology and even though it would be needed more time to produce a reliable simulation of it.

The first objective of this thesis was to introduce the resulting vectors from the LIF algorithm in the implemented network within OPNET and therefore demonstrate the accuracy of the algorithm, which could be a very interesting to step to carry on and it could be simple if a way of defining the input packets as vectors in OPNET.

After a deep understanding of OPNET and effective manipulation of its potentialities, as also proposed in the beginning of this thesis, it could be simulated a P2P streaming network using the NC algorithms developed in this thesis.

Appendix A

Here are set some of the MATLAB functions used for performing the needed calculations over the finite fields set for the implementation of the algorithms, as the vectors of the spanned finite field used in the implementations of the LIF algorithm, $\mathbb{F}_7^2$. It is also shown the packet flow of the NC Tutorial [7] within OPNET.

Function	Purpose
gfadd(a,b,p)	Adds two $\mathbb{F}_p$ polynomials, where *p* is a prime number. Variables *a* and *b* are row vectors that give the coefficients of the corresponding polynomials in order of ascending powers. Each coefficient is between *0* and *p-1*. If *a* and *b* are matrices of the same size, the function treats each row independently. (equal to c=(a+b) mod *p)*.
gfsub(a,b,p)	Calculates *a* minus *b*, where *a* and *b* represent polynomials over $\mathbb{F}_p$ and *p* is a prime number. *a* and *b* are row vectors that give the coefficients of the corresponding polynomials in order of ascending powers. Each coefficient is between *0* and *p-1*. As before, if *a* and *b* are matrices of the same size, the function treats each row independently.
gfmul(a,b,p)	Multiplies *a* and *b* in $\mathbb{F}_p$. Each entry of *a* and *b* is between *0* and *p-1*. *p* is a prime number. If a and b are matrices of the same size, the function treats each element independently.
gfdiv(b,a,p)	Divides *b* by *a* in $\mathbb{F}_p$ and returns the quotient. *p* is a prime number. As before, if *a* and *b* are matrices of the same size, the function treats each element independently. All entries of *b* and *a* are between 0 and *p-1*.

Table A.1 – MATLAB GF(p) operations functions. Adapted from [31].

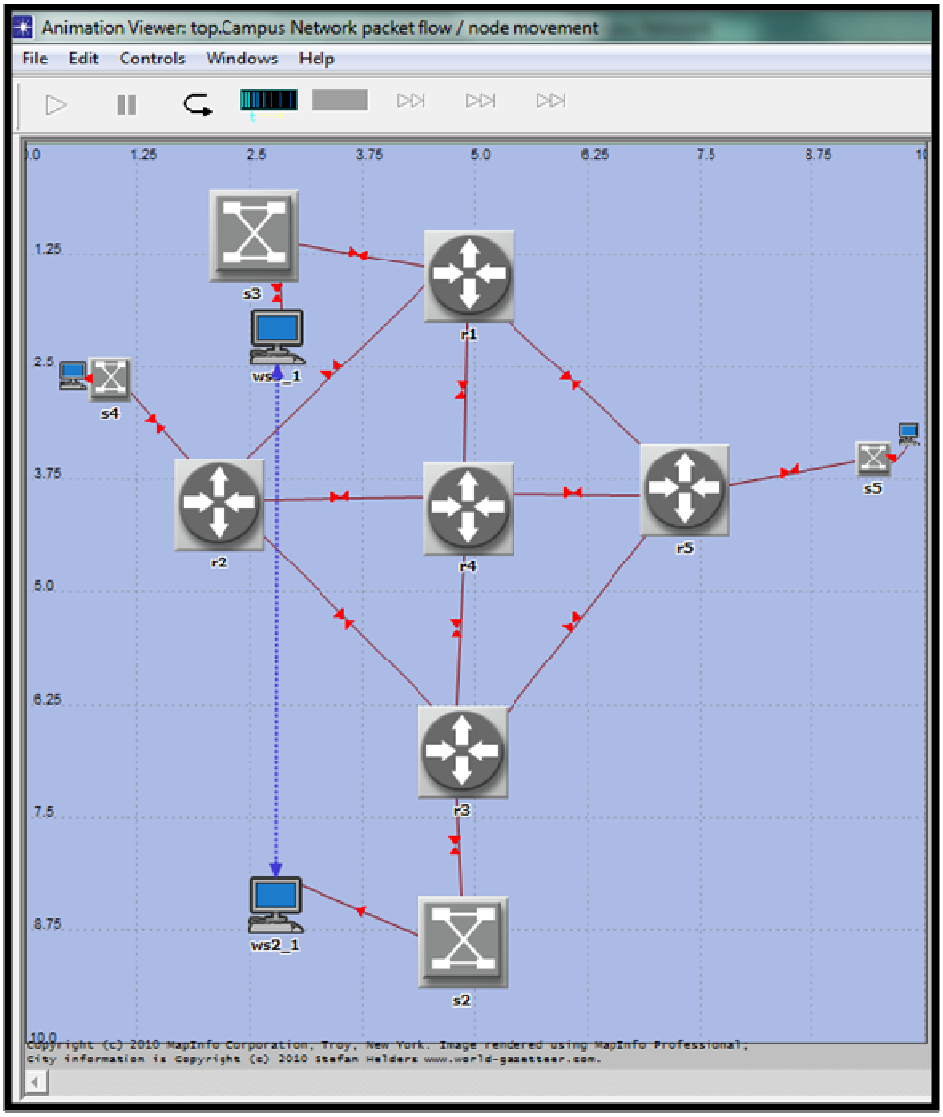

Figure A.1 – Packet flow of the NC tutorial network.

The vectors of the spanned finite field $\mathbb{F}_7^2$, used in the calculations of the algorithms are:

$$\mathbb{F}_7^2 \mapsto \begin{bmatrix} (0\ 0) \\ (0\ 1) \\ (0\ 2) \\ (0\ 3) \\ (0\ 4) \\ (0\ 5) \\ (0\ 6) \\ (1\ 0) \\ (1\ 1) \\ (1\ 2) \\ (1\ 3) \\ (1\ 4) \\ (1\ 5) \\ (1\ 6) \\ \vdots \\ (5\ 0) \\ (5\ 1) \\ (5\ 2) \\ (5\ 3) \\ (5\ 4) \\ (5\ 5) \\ (5\ 6) \\ (6\ 0) \\ (6\ 1) \\ (6\ 2) \\ (6\ 3) \\ (6\ 4) \\ (6\ 5) \\ (6\ 6) \end{bmatrix}$$

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, R. W. Yeung, *"Network information flow"*, IEEE Trans. on Information Theory, vol. 46, pp. 1204-1216, July 2000.

[2] C. Fragouli, J. Widmer, *"Network coding: an instant primer"* ACM SIGCOMM Computer Communication Review, vol. 36, pp. 63--68, 2006.

[3] C. Fragoulli, E. Soljanin, *"Network Coding Fundamentals",* Foundations and Trends in Networking, Vol. 2, No. 1, 2007.

[4] P. Chou and Y. & Jain, K. Wu, *"Practical Network Coding",* Proceedings of the annual Allerton Conference on Communication Control and Computing, vol. 41, pp. 40-49, 2003.

[5] http://www.networkcoding.info/

[6] http://www.ee.washington.edu/research/funlab/network_coding/index.html

[7] http://www.ee.washington.edu/research/funlab/network_coding/opnet_nc.pdf

[8] N. Cai, R. W. Yeung, *"Secure network coding"* ISIT, 2002.

[9] N.Thomos, P. Frossard, *"Network Coding: from theory to media streaming"* June 2009.

[10] P. A. Chou, *"Practical Network Coding for the Internet and Wireless Networks"* Globecom Tutorial, December 3, 2004.

[11] J. L. Van Wyk, , A. S. J. Helberg, and M. J. Grobler, *"Comparing the implementations of Network Coding on different OSI layers",* SATNAC, Royal, Swaziland, 2009.

[12] M, Wang, B. Li, *"Network Coding in Live Peer-to-Peer Streaming"* IEEE Transactions On Multimedia, Vol. 9, No. 8, December 2007.

[13] R. Koetter, "*An Algebraic Approach to Network Coding*" IEEE/ACM Transactions On Networking, Vol. 11, No. 5, October 2003.

[14] H. Wang, J. Liang, C.C. J. Kuo, *"Overview of Robust Video Streaming with Network Coding",* Journal of Information Hiding and Multimedia Signal Processing, Vol. 1, No. 1, January 2010.

[15] S. Jaggi, P. Sanders, P. A. Chou, M. Efros, S. Egner, K. Jain, L. M. G. M. Tolhuizen, *"Polynomial Time Algorithms for Multicast Network Code Construction"*, IEEE Transactions on Information Theory, Vol. 51, No. 6, June 2005.

[16] J. K. Sundararajan, M. Médard, M. Kim, A. Eryilmaz, D. Shah, R. Koetter, "*Network Coding in a Multicast Switch*", Proceedings of IEEE. Infocom, 2007.

[17] M. Langberg, A. Sprintson, J. Bruck, "*Network Coding: A Computational Perspective"*, IEEE Transactions On Information Theory, Vol. 55, No. 1, January 2009.

[18] J. Barros, *"Mixing Packets: PROS and CONS of Network Coding",* The 11th International Symposium on Wireless Personal Multimedia Communications (WPMC'08).

[19] http://www.dcc.fc.up.pt/~neco/Docs/report.pdf

[20] M. Langberg, A. Sprintson, J. Bruck, *"The Encoding Complexity of Network Coding",* IEEE Transactions On Information Theory, Vol. 52, No. 6, June 2006

[21] S. Y. R. Li, R. W. Yeung, N. Cai, *"Linear Network Coding",* IEEE Transactions On Information Theory, Vol. 49, No. 2, February 2003

[22] L. R. Ford Jr., D. R. Fulkerson, *"Maximal flow through a network,"* Canadian Journal of Mathematics, Vol. 8, pp. 399–404, 1956.

[23] K. Menger, "Zur allgemeinen Kurventheorie," Fundamenta Mathematicae, Vol. 10, pp. 95–115, 1927.

[24] A.J. González, D. Rodríguez, J. López, F. I. Rillo, J. Alcober, "*Streaming P2P robusto en redes Ad-hoc utilizando información social",* Jornadas De Ingeniería Telemática, 2009.

[25] http://cs.nju.edu.cn/wuxb/NC.pdf

[26] T. Ho, R. Koetter, M. Medard, D. R. Karger, M. Effros, *"The Benefits of Coding over Routing in a Randomized Setting",* IEEE International Symposium on Information Theory, 2003.

[27] Á. Barbero, Ø. Ytrehus, "*Knotwork coding"*, Proceedings Information Theory and Applications Workshop, San Diego, CA, Feb. 6-10, 2006.

[28] http://www.cse.psu.edu/~venkates/index_files/opnet_tips.html

[29] J. Dumas, T. Gautier, C. Pernet, *"Finite field linear algebra subroutines"*, Proc. Int. Symp. Symbolic and Algebraic Computation(ISSAC), Lille, France, pp.63–74, Jul. 2002.

[30] V. Dham, "*Link establishment in ad hoc networks using smart antennas*." Master thesis, Virginia Polytechnic Institute and State University, 2003.

[31] K. Imamura, *"A method for computing addition tables in GF(p ^{n})"*, *IEEE* Transactions on Information Theory, Vol. IT-26, No. 3, pp. 367–369, May 1980.

[32] S. Katti, D. Katabi, *"Wireless Network Coding: Opportunities and Challenges"*, MILCOM, 2007

[33] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, J. Crowcroft, "*Xors in the air: practical wireless network coding."*, SIGCOMM. Pisa, Italy: ACM, September 2006, pp. 24 –254.

[34] Online Documentation, OPNET Modeler 12.0.A.

[35] http://www.opnet.com/

[36] Product Help, Matlab 7.9.0

[37] www.mathworks.com/

[38] http://scholar.google.com/advanced_scholar_search

[39] http://www.gliffy.com/gliffy/#

[40] http://www.ams.jhu.edu/~ers/matgraph/