Self-adapting Linear Network Coding Emulation

Nuno B. Coelho, Francisco A. Monteiro, Rui J. Lopes
Instituto de Telecomunicações
and
Dep. of Information Science and Technology, ISCTE - Instituto Universitário de Lisboa
Lisbon, Portugal
francisco.monteiro@lx.it.pt , rui.lopes@iscte-iul.pt

***Abstract*— Linear network coding (LNC) introduced a new paradigm for routing data across networks where the transported packets are not the original information packets but rather linear combinations of the original packets. This is known to allow a more efficient use of the network resources. With LNC based on systematic network codes, the network's capacity is chiefly defined by the rank of the end-to-end equivalent transfer matrix. This paper presents a network-layer emulator based on the object-oriented programming paradigm, which allows, for any network topology, to have the network's capacity self-optimized by adapting the coding operations performed at each node, depending on the impact the changes have on the global rank of the global transfer matrix, and on a metric related to the use of network's resources.**

Keywords— Linear network coding; Software defined networks, Network resilience; Rank metric; Java emulation.

I. INTRODUCTION

Network coding (NC) was first presented as a method for error correction in [1], which is deemed to be the first paper on the topic. It was then shown how NC allows transmitting more information across a network in comparison to traditional routing [2]. In general, a well-designed NC method guarantees both an efficient use of the network's resources and robust end-to-end connections in case of link failures [3].

Both wireless and wired packet-based network may have different topologies and different types of nodes [3] [4], but are usually capable of establishing several different paths between the source and destination nodes by setting different connections at intermediate nodes. These intermediate nodes may be responsible for making their own forwarding decisions, based on the global or local knowledge they have of the network, or they can be controlled by some central network manager. This latter approach has gained relevance in the context of software defined networks (SDN). The overall transfer matrix of the network can be known, corresponding to the so-called coherent model or unknown even to a central processor, called the non-coherent model (see, e.g., Kschischang's preface in [5]).

In the most common case of binary linear NC (LNC), the intermediate nodes always combine the packets from different sources by applying XOR functions to sets of packets. Research on fundamentals and applications of binary and non-binary network coding has been mounting and recent comprehensive results and surveys can be found in [5] [6]. It should be noted that NC ideas are not limited to the packet transmission layer and its core idea have also been extended to the physical layer of wireless networks (e.g., [7] [8]).

The work presented in this paper was initially inspired by the the protection scheme for multi-hop wireless networks proposed in [9]. That protection scheme was tested with one and two intermediate node failures and quantified the quality of service (QoS) by means of parameters such as packet loss ratio (PLR) and latency. In [10], the authors used NC to introduce a protection scheme against single and multiple link failures, recovering a second copy of each data unit transmitted "automatically" without rerouting data or without failure detection. In [11], the authors proposed a general method to design the transmission protocol with binary physical-layer network coding (PLNC). They examined several proposed protocols in terms of energy consumption, error rate, and throughput performance and decoding strategy.

This paper presents a LNC emulator which allows the study and assessment of different LNC strategies, different network types (either wired or wireless), under any topology specified by the user. The QoS metrics considered in the analysis are the PLR and latency. The main contributions of this work are its grounding on fundamental properties of NC (e.g., verification of conditions for viability); and on the other hand its flexibility (e.g., the coding coefficients can be computed by the emulator or provided by the user and obtained via any other means). This emulator was developed using multiple *threads* in Java and provides a more flexible alternative to other simulators such as the one in [12] (written in Python) or the one in [13] (written in C++). The presented emulator allows simulating network coding with any defined network topology and implements a self-adaptation mechanism for the network coding operation carried at the nodes. The emulator was tested with the same cases analyzed in [9].

The paper is organized as follows. The next section introduces LNC in packet networks. Section III describes the developed emulator and its *a priori* requirements. Section IV describes the network topology considered. Section V describes the operations held at the different nodes to combine the incoming packets. Section VI shows how a viable network coded network is always guaranteed after the self-adaptation process of the network codes that take place at each node. In Section VII a central control unit is added (a "genie" or "genius"). Section VIII presents results obtained with the network emulator, as well as the conclusions.

II. A Brief Introduction to Linear Network Coding

LNC increases the throughput of a network from a source node to a destination node due to a more efficient use of the several physical paths between them. Additionally, the robustness to packet losses and link failures is increased given that the information packets that did not arrive at their destination can be inferred from the linear combinations of the coded packets that have arrived. Also, LNC brings more security and complexity to the network [14]. However, the use of LNC requires transmitting additional information along with the messages, notably the information of which packets have been combined to form each coded packet [15]. When doing this, there is no need to store additional information in the nodes of the network. This extra information can be easily placed in packet headers, which in this paper are called "transfer vectors". The existence of headers is common in standard protocols like the TCP protocol [16].

A. Operations in LNC and coded packet structure

In LNC the operations at the nodes are constrained to be linear over a finite integer field with q symbols in the alphabet, i.e., a Galois field denoted as $\mathbb{F}_q$ or GF(q) [17], [4], [5]. In this work a binary field is considered, and therefore both the transmitted packets and the coefficients of the linear combinations are taken from GF(2). All operations (namely addition, division, multiplication, subtraction, and Gauss-Jordan elimination) are defined over GF(2) [11] [18].

Of chief practical importance is the structure of the packets that are exchanged between nodes, and how they convey both the coded packets (in the payload) and the control information about how the packets have been coded. That is, packets arriving at the j^{th} input of node N result from the concatenation of a transfer row vector, $\mathbf{h}_{N,j}^{(in)}$, and its corresponding coded packet, $\mathbf{x}_{N,j}^{(in)}$, having the format $\left[\mathbf{h}_{N,j}^{(in)}\middle|\mathbf{x}_{N,j}^{(in)}\right]$. One of the fundamental processes of LNC is expressed by $\left[\mathbf{h}_{N,j}^{(in)}\middle|\mathbf{x}_{N,j}^{(in)}\right] = \left[\mathbf{h}_{N,j}^{(in)}\middle|\mathbf{h}_{N,j}^{(in)}\boldsymbol{X}_s\right]$, where $\mathbf{X}_s$ is a matrix having in its rows the original source packets. Naturally, the number of columns in $\mathbf{h}_{N,j}^{(in)}$ must match the number of rows in $\mathbf{X}_s$. The stacking of all row vectors $\mathbf{h}_{N,j}^{(in)}$ (i.e., the transfer vectors) creates matrix $\mathbf{H}_N^{(in)}$, and the stacking of the row vectors $\mathbf{x}_{N,j}^{(in)}$ (i.e., the coded packets arriving at node N) generate matrix $\mathbf{X}_N^{(in)}$.

At each node one has packets in $\mathbf{X}_s$, which correspond to a burst or "generation". Both the size of the packets (number of columns in $\mathbf{X}_s$), and the size of each "generation" (number of rows in $\mathbf{X}_s$) are system parameters. It should be noted that a row vector $\mathbf{h}_{N,j}^{(in)}$ describes "the memory" of all the linear operations performed at each visited node and that transformed the original source packets into the coded packet arriving at the j^{th} input of node N. In fact, this is a direct result of using systematic network codes [19], hence, the left part (control information) of the packet tracks its previous path and directly reveals what is being combined in the coded message part of the packet (payload).

Similarly, at the output of the i^{th} output of node N one will find packets of the form:

$$\left[\mathbf{h}_{N,i}^{(out)}\middle|\mathbf{x}_{N,i}^{(out)}\right] = \left[\mathbf{h}_{N,i}^{(out)}\middle|\mathbf{h}_{N,i}^{(out)}\boldsymbol{X}_s\right] \quad (1)$$

B. Types of Nodes

One now describes how the different elements in a network process coded packets, starting with the intermediate nodes – or simply network nodes (typically routers if the network layer is considered). In traditional packet networks, the role of an intermediate node is to forward received packets to another node "closer" to the destination. With NC, the intermediate node firstly encodes received packets and later forwards the coded packets to another node "closer" to the destination.

Similarly to what has been expressed for the input, 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

$$\left[\mathbf{H}_N^{(out)}\middle|\mathbf{X}_N^{(out)}\right] = \mathbf{C}_N\left[\mathbf{H}_N^{(in)}\middle|\mathbf{X}_N^{(in)}\right], \quad (2)$$

where $\mathbf{C}_N$ is the connection matrix of node N, which describes the coding (or combining) process performed at that intermediate node. Note that after the coding takes place at a node, one has the output transfer matrix $\mathbf{H}_N^{(out)} = \mathbf{C}_N\,\mathbf{H}_N^{(in)}$ and the new set of vectors $\mathbf{X}_N^{(out)} = \mathbf{C}_N\,\mathbf{X}_N^{(in)} = \mathbf{C}_N\,(\mathbf{H}_N^{(in)}\mathbf{X}_s)$. The last equality denotes the fact that the header $\mathbf{H}_N^{(in)}$ contains the information of the accumulated network coding operations that the arrived packets went through from the source until the present node to which they are arriving. Moreover, the columns of $\mathbf{C}_N$ are associated to the node's logical inputs and the rows are associated to the node's logical outputs. The inputs and outputs considered at each node do not have to correspond necessarily to physical inputs or outputs. Actually, by considering logical inputs and outputs enables that coded packets can be sent over physical links at different rates. The practical implementation of such virtual inputs and outputs is discussed in section VIII.

Let us now consider a simple example where $\mathbf{C}_N$ describes the LNC processing at node with three logical inputs and two logical outputs with $\mathbf{C}_N = \begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 1\end{bmatrix}$ over $\mathbb{F}_2$. With some simplification of the notation, assume that a node N received from source s_i, at each of its three inputs, 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 (note that this set of headers is the particular case that always happens at the first node of a path), all belonging to the same burst or to the same "generation". In particular, the header of the second combined packet output becomes:

$$\mathbf{h}_{N,2}^{(out)} = [\,h_{x1}(1)\oplus h_{x3}(1)\,,\, h_{x1}(2)\oplus h_{x3}(2),\, h_{x1}(3)\oplus h_{x3}(3)] \quad (3)$$

and, in general

$$\mathbf{H}_N^{(out)} = \mathbf{C}_N\,\mathbf{H}_N^{(in)} = \begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 1\end{bmatrix} \quad (4)$$

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

$$= \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \quad (5)$$

On the other hand, source nodes will be responsible for generating and sending messages to a certain destination in the network. 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. To encode the packets, the source node uses linear independent vectors (also called a *linear lifting* [3], [20]). If the source s wants to send packets p_1, p_2 and p_3 over $\mathbb{F}_2$, the initially transmitted packets can be stacked and create

$$\mathbf{X}_s = [\mathbf{H}_s|\mathbf{P}] = \left[\begin{array}{ccc|c} 1 & 0 & 0 & p_1 \\ 0 & 1 & 0 & p_2 \\ 0 & 0 & 1 & p_3 \end{array}\right], \quad (6)$$

where the transfer matrix at the source is the identity matrix (c.f. the previously given example).

Finally, the destination node is responsible for decoding received packets, as described in Algorithm 1. A fundamental fact influencing the reliability of the system is that in order for the destination to be able to decode the set of packets that arrive to it the rank of the global transfer matrix needs to be equal or larger than the number of source packets in one particular generation. 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, over GF(2), the system $\mathbf{Y} = \mathbf{HP}$ for the **P** matrix**.** This involves performing one matrix inversion over GF(2) at the destination d:

$$\left(\mathbf{H}_d^{(in)}\right)^{-1}\left[\mathbf{H}_d^{(in)}\middle|\, \mathbf{X}_d^{(in)}\right]. \quad (7)$$

Algorithm 1: Algorithm at the destination 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 in buffers
out ← ∅
add packet p to buffer of link l
for_each b of inBuffers do
   for_each p2 packet of b do
      if there is a bucket created for p2 source
      node, with p2 generation number and p2 destiny
      node then
         if b increases rank then
            add p2 to b
         end
      else
         create new bucket for p2
      end
   end
end
```

It becomes clear from both the last expression and from Algorithm I that the rank of the matrix to be inverted is of capital importance for a successful decoding, and thus to make a decision about the viability of the network structure and connection matrices. This will be explored further in section VI, where a process for designing LNC viable networks is described.

III. Network Emulation

Based on the concepts laid out in the previous section, an emulator was developed in Java (named "Net Genius") using the object-oriented paradigm. In the emulator each node in the network has the same processing functionalities and reacts to inputs, with all nodes working in parallel, emulating the routing of data by means of multiple threads. Each node is thus emulated by a thread and each thread is independent from the others. The emulator has two operating modes: a basic mode and an advanced mode.

In the basic mode, the network structure and the operation at each node occurs according to the defined physical and logical links and predefined connection matrices, whether set manually or loaded from a preset configuration. Each node is only aware of neighbor nodes to which it is connected to, corresponding to a decentralized network. In this mode via probing packets it is also possible to check if packets can be decoded (that is, the network viability).

Conversely, in the advanced mode, only the network structure is predefined, the operation at each node, i.e. its connection matrix, is centrally defined by a system entity (a "genie" or "genius") that controls the network.

A. Flexible Emulation of NC

The emulator was made flexible enough to either implement traditional routing or NC. The user only needs to define the network topology, and then the linear model defined in the previous section suffices to emulate the routing using NC. The user has the option of defining the physical and logical links between nodes either by manually configuring the network nodes in the emulator setup or by using the emulator console's 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 depending on the physical connections, whether it is a manual configuration or a pre-set configuration. Unlike the emulators in [12] and [13], the presented emulator allows the user to define the coding matrixes are each node, as one would expect in a truly SDN context. For example in [12] the coefficients are sampled from a uniform distribution. The emulator then checks if the defined network topology and the NC operations lead to a viable network, by checking the rank of the end-to-end transfer matrix.

B. Emulation of the different nodes

In the following, one describes how the different nodes in of a LNC enabled network were emulated by means of a running example. Assume that packet p, belonging to the second generation of source s, arrives at destination node g. When g receives the packet, it will check if there is a bucket created for that generation and source. If that bucket is already created, p will be added to the existing bucket only if it "adds value" to the existing bucket, i.e., if the bucket rank increases or reaches the desired rank. Each node will have several buckets that will store the packets of different sources or generation.

In the example in Figure 1, the intermediate node r_1 is connected to two sources and to three other intermediate nodes; it has two IN buffers to store the received packets and three OUT buffers to store the computed packets to be sent.

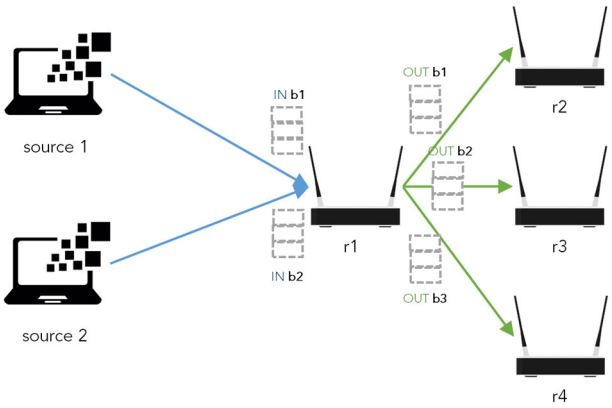

Figure 1 - Buffers of the intermediate node.

C. *Packet structure*

The packed packet structure proposed and used in the emulator is shown in Figure 2. This structure enables the linear model presented in Section II to be implemented in real network. The fields in the packer header contain control information that is needed for coding and decoding the network-coded data field, which is the message itself.

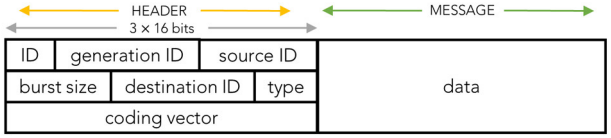

Figure 2 - Packet structure.

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 s_1, its ID will be "s_{1n}" where n is a sequence number within a certain generation from that source node. The generation ID identifies the generation number of the packet and this generation number is associated with the burst size (also named the "generation size"). For example, in a burst of size four, there will be four packets belonging to the same generation. 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. Only two values can be assigned to the type field: [INFORMATION, PROBING]. These values indicate if a packet contains information (data packets), or if it is one of the so-called "probing packets" that are used to validate the NC transmission, i.e., to check whether the messages can be correctly decoded.

IV. Preset Network Topology

Hereafter one assumes the particular case of the network depicted in Figure 3, with one source node, six intermediate nodes and one destination (or terminal), which corresponds to the preset network present in the emulator. Assume that node r_1 receives packets p_1, p_2, p_3 and p_4 from s_1. The transfer vectors (in $\mathbb{F}_2$) are [100], [011], [101], and [110] respectively. and therefore, the transfer matrix of r_1 is

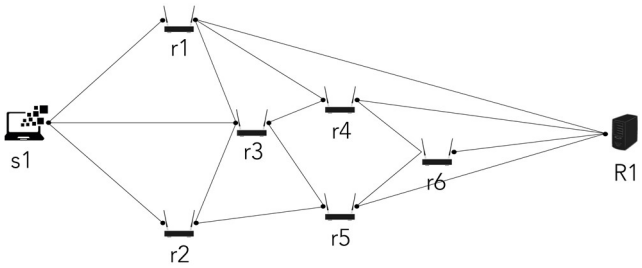

Figure 3 - Network model.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}. \quad (8)$$

In order to correctly decode the packets received, r_1 cannot have a rank deficient **H** matrix, i.e., it must receive a minimum of m linearly independent transfer vectors from each generation [18], where m corresponds to the number of packets of a generation, i.e., the burst size. In this example, the source s_1 sends three packets and destination r_1 receives four packets. The required rank of **H**, so that r_1 is able to correctly decode the source packets, is 3 (as 3 packets have been coded and sent).

V. Coding and Decoding

The version of the described emulator only combines packets originating from the same source node. The idea of combining information from different sources would pose a number of challenges for buffer management, however this proposal does not undermine the extension of this LNC process to the more general scenario. Note that instead of having a fixed transfer matrix **H**, the intermediate nodes and the destination nodes have a dynamic **H** matrix. Each packet carries a transfer vector in the packet header and this transfer vector becomes a row of **H** at each intermediate node. At each output i of a node, one gets a linear combination of packets received by that node, defined by the i-th row of the connection matrix **C**. Only packets with the same generation number and from the same source are encoded together. In the example in Figure 4, r_3 will always combine packets from r_1 and r_2 and send the encoded packet to r_4 and r_5. In the general case instead of r_3 having just one connection matrix, there will be n connection matrices, where n is the number of source nodes that are connected to the node.

Without any loss of generality, we assume in the remainder of this paper that there is only one source. As one may see in Figure 4, what is sent to r_4 differs from what is sent to r_5. 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 $x_4 = x_1 \oplus x_2$. It should be highlighted that the inputs and outputs are the *logical* links connected to the node and *not* physical links. Considering that two physical links exist, one with capacity C and another with capacity $2C$, from the NC perspective one accounts for the existence of three logical links. Initially all the elements of **C** are set to 1, that is, by default all the outputs of a node replicate the XOR of all the received packets by that node.

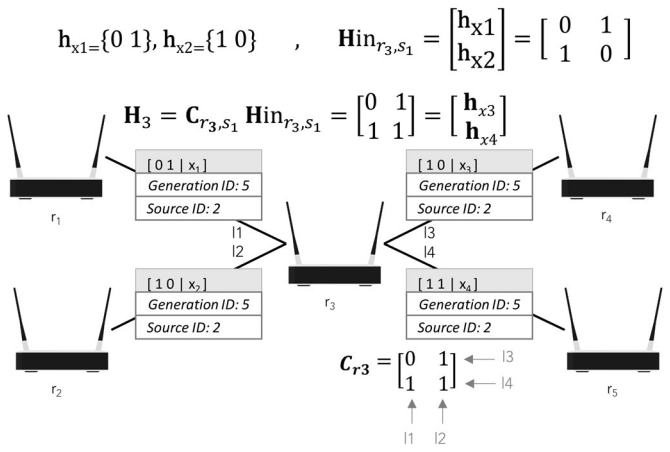

Figure 4 - Connection matrices of node r_3.

VI. Rank Metric for a Viable Network

When a user sets up a determined configuration, by manually configuring the network physical or logical communication links and choosing the basic mode, one needs to guarantee that the network is viable, i.e., that it guarantees a certain level of reliability under certain network conditions. In our emulator this process is based on the exchange of probe packets and verification of a set of criteria. Probe packets are forwarded at each node according to connection matrix **C.** The two criteria that must be satisfied are:

1. *Connection criterion*: each destination node must be connected to at least one source node (this criterion guarantees that at least one probe packet will be received by each destination node);
2. *Linear independence criterion*: each destination node must have an **H** matrix whose rank is equal or greater than the burst size (i.e., the size of the generation).

To guarantee that the first criterion is verified, the recursive Algorithm 2 was created. 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. This procedure is applied by all destination nodes of the network. If a destination has zero connections to a source node, the user is notified and the same happens if all destination nodes have zero connections to a source node. In short, the algorithm starts from a destination node and searches through the connected links for source nodes, ensuring that the destination is connected to at least one source node. A node that is not physically connected to another node has a null matrix as a connection matrix, i.e., **C** = []. So, the algorithm can perform this verification recursively throughout the connection matrices.

Algorithm 3 ensures the second viability criterion. To that end, at the beginning of the emulation probing packets are sent with the objective of checking if the network is viable in terms of correctly decoding the message sent by the source node. When the destination receives all the probing packets it will check if the transfer matrix reaches the desired rank, i.e., if is not rank deficient. Note that the minimum desired rank of **H** is equal to the burst size (i.e., the size of the generation).

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

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

_____________________________________________
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 ← ∅
   for each b of inBuffers do
      if n is an intermediate node then
         for_each link of node n 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
                  coming from b, searching on n2
               end
            end
         end
      else if n is a source node then
         add n2 to the sources found
      end
      end
   end
```

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
```

VII. Self-adaptation of Network Coding

In the advanced mode there is an entity (called "genius") that has full knowledge of the network. This means that the process of checking the required linear independence is done quicker than in the basic mode, because there is no need for sending the probing packets. In fact, in advanced mode, the genius does more than checking linear independence. Instead of only checking if the system is linearly independent as in the basic mode, the genius attempts to gradually improve the network by changing the appropriate connection matrices. By 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. The genius uses a recursive algorithm to test and change several combinations of the network physical connections. Each time the user starts the emulation for the same network, the genius will execute this algorithm, so that the network's configuration may change; this allows the user to test the network with different configurations until the genius finds the best possible configuration. In order for the genius to accept a new configuration, that particular configuration must have a better rating than the last saved configuration. The configuration rating, defined as CRa, is the acceptance criteria considered, and is calculated in the following manner:

$$CRa = w_R \times \frac{Actual\ Rank}{Max\ Rank} + w_C \times \frac{Actual\ Cost}{Max\ Cost} \quad (9)$$

This algorithm is based on the weights w_R and w_C, corresponding to the overall matrix rank and to a network resources' cost. The latter is the total number of links within the network, i.e., the number of links that are being used by all nodes (each link connects two nodes). The former is the metric that assures that packet decoding is possible at the destination node. By considering these two metrics, one obtains a fitness function that can be used by different methods to set the connection matrices and thus build a self-adapting network (e.g., simulated annealing, or gradient descent methods). Packet loss is important in scenarios where LNC is used for improving the network reliability. In order to assess LNC configurations in scenarios with packet losses the user has the option of choosing the link error probability for each emulation, independently of the emulation mode. Being P_k the packet loss probability in all the links, and s the number of jumps (hops) between the source and the destination along a certain path, the packet loss probability, P_e, over that path, is given by $P_e=1-(1-P_k)^s$. In the examples of Section VIII one considers scenarios with three different values for the packet loss probability, P_k: no packet loss, 2% and 10%.

VIII. Example Results and Conclusions

In each emulation and for each node, the emulator presents the number of buckets expired, the number of buckets computed, the number of packets received, and the number of packets sent. In the following are presented, for each source, the number of packets sent and the number of generations. Nice different cases were emulated for the network, and are listed in Figure 3. The coding method and the packed loss probability (P_k) used in each case are listed in Table I. In all cases w_R and w_C were set to 0.7 and 0.3, respectively. In order to assess the performance of each case, one uses the packet loss ratio (PLR) and a latency metric. PLR corresponds to the ratio of total number of packets successfully received at the destination node over the total number of packets sent by all sources. Latency is calculated as the maximum sum of the accumulated propagation delay, switching delay, and transmission delay over all the paths between source and destination.

TABLE I – Emulated scenarios.

Case	Method	P_k
1	No LNC	0 %
2	No LNC	2 %
3	No LNC	10 %
4	Distributed LNC	0 %
5	Distributed LNC	2 %
6	Distributed LNC	10 %
7	Centralized LNC	0 %
8	Centralized LNC	2 %
9	Centralized LNC	10 %

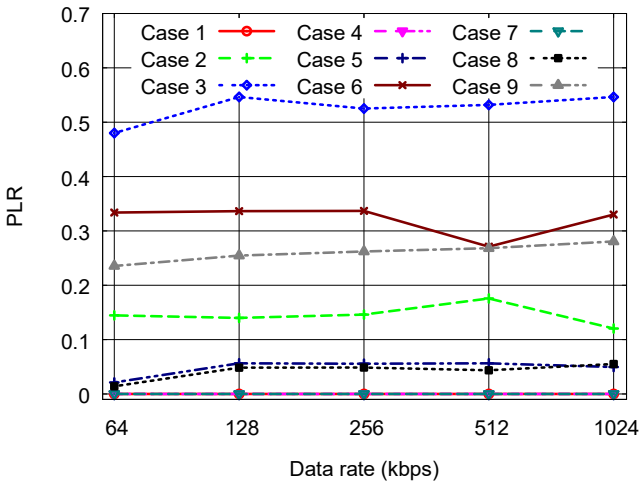

Figure 5 - PLR results for cases 1-9 for the network presented in section IV.

Figure 5 shows the results of PLR for the emulation of the network presented in Figure 3. In the cases with NC, the PLR is always below 40%. When there are no errors in the links, the PLR is consequently zero (overlapping curves at the bottom of the graph), and it increases for larger error probabilities. In these cases of non-zero link error probabilities, the traditional approach presents worst PLR results than NC. This was to be expected given that in NC, when a packet is lost, the information may be recovered from a set of linear combinations of native packets. This contrasts with the traditional approach where there are no coded packets, and therefore, when a packet is lost, there is no way of recovering that packet unless it is retransmitted.

Between centralized and distributed LNC one cannot find significant differences in the results in Figure 5 (although with higher error probabilities in the links the centralized approach was seen to perform slight better). The paper shows that lossy-networks benefit from a network coding approach due to the resilience provided by the linear combinations of packets. In spite of a higher implementation complexity, the centralized network coding approach is the one that achieves the best QoS metrics in all the emulated scenarios. It was also seen in the emulation results that NC performs worse than the traditional routing in terms of latency and jitter (defined as the maximum variation between the arrival time of packets), as shown in Figures 6 and 7. This is due to the queueing and processing time at the nodes. In the case of NC, the nodes have to wait and queue all the required packet from each generation and compute the GF(2) LNC operations before forwarding them, so the

processing time at the nodes is higher than the one taken by the traditional routing approach. Although more resilient to failure than the traditional approach, and besides the slight increase of decoding complexity, the most significant price to pay when using NC is latency.

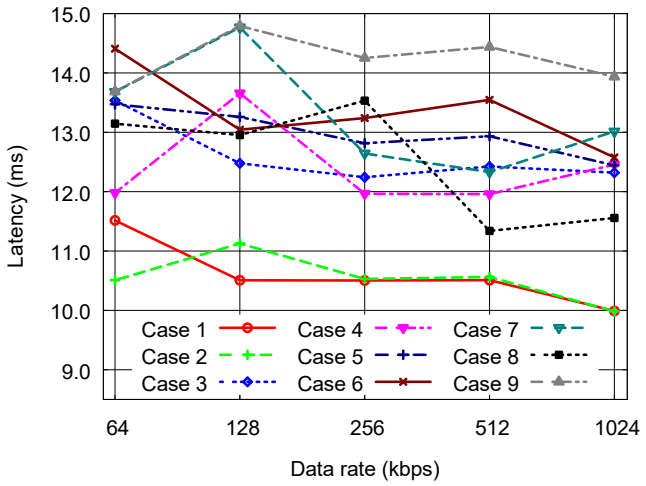

Figure 6 - Latency results for cases 1-9 for the network presented in section IV.

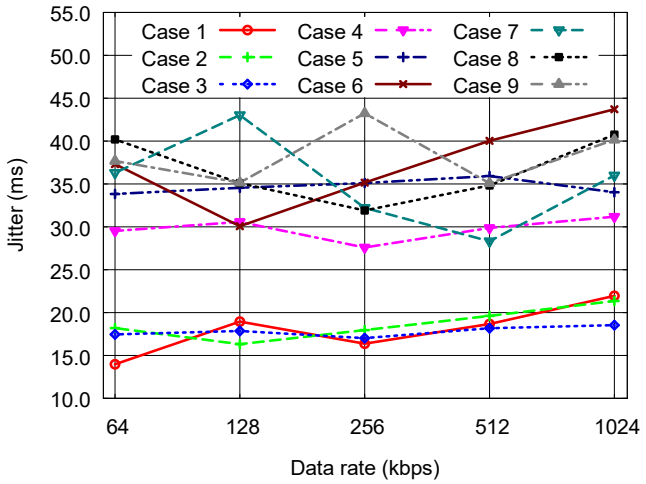

Figure 7 - Jitter results for cases 1-9 for the network presented in section IV.

A natural extension to this work would be the inclusion of error correction mechanisms. Testing other network 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 an eavesdropper, who tries to decode some information from the network-coded packets.

ACKNOWLEDGMENT

This work was funded by FCT (Foundation for Science and Technology) and Instituto de Telecomunicações under project UID/EEA/50008/2019. Francisco Monteiro is also grateful to the opportunities granted by the European COST Action IC1104 “Random Network Coding and Designs over GF(*q*)”.

REFERENCES

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

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

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

[4] Y. Qin, Network Coding at Different Layers in Wireless Networks, Germany: Springer, 2016.

[5] M. Greferath, M. Pavčević, N. Silberstein and M. Vázquez-Castro, Network Coding and Subspace Designs, Springer, 2018.

[6] F. A. Monteiro, A. Burr, I. Chatzigeorgiou, C. Hollanti, I. Krikidis, H. Seferoglu and V. Skachek, Special issue on network coding, Springer, 2017, pp. DOI 10.1186/s13634-017-0463-2.

[7] Katti, S; Rahul, H; Hu, W; Katabi, D; Medard, M; Crowcroft, J., "XORs in the air: practical wireless network coding," Pisa, Italy, October 2006.

[8] J. S. Lemos, F. A. Monteiro, I. Sousa and F. E. Ferreira, "Efficient message exchange protocols exploiting state-of-the-art PHY layer," *EURASIP Journal on Wireless Communications and Networking,* no. 2017:92, pp. DOI 10.1186/s13638-017-0850-2, May 2017.

[9] B. Saeed, P. Rengaraju, C. Lung, T. Kunz and A. Srinivasan, "QoS and protection of wireless relay nodes failure using network coding," in *Proc of International Symposium on Network Coding (NetCod)*, Beijing, China, 2011.

[10] A. Kamal, A. Ramamoorthy, L. Long and S. Li, "Overlay protection against link failures using network coding," *IEEE/ACM Transactions on Networking,* vol. 19, no. 4, pp. 1071-1084, 2011.

[11] R. Chang, S.-J. Lin and W.-H. Chung, "Transmission protocol design for binary physical network coded multi-way relay networks," in *IEEE 79th Vehicular Technology Conference (VTC Spring)*, Seoul, Korea, May, 2014.

[12] D. Ferreira, L. L. Lima and J. Barros, "NECO: NEtwork COding Simulator," in *Proc Inter. Conf. on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTOOLS)*, Rome, Italy, March, 2009.

[13] N. Marcano, M. V. Pedersen, P. Vingelmann, J. Heid, D. E. Lucani e F. Fitzek, “Getting Kodo: Network Coding for the ns-3 Simulator (WSN3),” em *2016 ACM Workshop on ns-3*, Seattle, WA, USA, 2016.

[14] C. Fragouli and E. Soljanin, "Introduction," in *Network Coding Fundamentals*, Foundation and Trends® in Networking, 2007, pp. 1-9.

[15] M. Médard and A. Springtson, "Linear network coding," in *Network Coding: Fundamentals and Applications*, Waltham, MA, Academic Press, 2011.

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

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

[18] J. Heide, M. V. Pedersen, F. H. Fitzek and M. Médard, "On code parameters and coding vector representation for practical RLNC," in *Proc. of IEEE Inter. Conf. on Comm. (ICC)*, Kyoto, Japan, 2011.

[19] 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.

[20] M. Bossert e E. M. Gabidulin, “One family of algebraic codes for network coding,” em *Proc. IEEE International Symposium on Information Theory*, Seoul, Korea, June, 2009.