Improving the decoding efficiency of Matrix Network Coding

TG van Dyk

orcid.org/ 0000-0002-8042-1934

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North West University

Supervisor: Prof ASJ Helberg

Co-supervisor: Dr M Ferreira

Graduation: July 2020

Student number: 24973742

Acknowledgments

I would like to acknowledge and thank those in my life that have made it possible for me to finish this dissertation and without whom I would not be the person I am today:

- My Lord and Saviour Jesus Christ through Whom I had the strength and ability to finish this dissertation. Thank You for the opportunity You gave me through this time to draw nearer to You. I owe all I am to You.
- My supervisor Professor Helberg, thank you for always being encouraging and bursting with new ideas everyday. Especially thank you for always being positive and seeing opportunity in all my doubts.
- My co-supervisor Doctor Melvin Ferreira, thank you for encouraging me to undertake these studies and for your involvement and dedication throughout my studies.
- Thank you to my mother, Mariana du Plessis, for supporting me through everything I do. You are a true role model. Thank you for all the sacrifices you have made for us. Also thank you to the rest of my family, my grandparents, brother and step-father, for your unconditional love.
- My friends, in particular, Ruan Erlank for the final encouragement and support I needed to finish this dissertation. Also thank you to Jacques, Dewald and Eljeanie for going through various parts of this journey with me.
- The TeleNet research group and all my other fellow research students for encouragement, advice and lunches with endless discussions, debates and laughs.
- The Telkom Centre of Excellence for financing my studies. A special thank you to Mr Gys Booysen for periodic encouragement and advice.

Abstract

To reach the maximum flow capacity of a network, network coding can be used. There are various implementations of network coding. The most widely used method of network coding is Random Linear Network Coding (RLNC). RLNC, however, is quite susceptible to network errors. Due to this drawback, Matrix Network Coding (MNC) was developed by Kim, et al. in 2011.

MNC improves on RLNC since it has better error-correcting capabilities. However, since MNC uses matrices for encoding rather than coefficients, it adds complexity to the system. Decoding is also more complex for MNC since the *G*-matrix is larger and not always invertible, which is necessary for decoding.

We focused on methods to improve the decodability of MNC by improving on the invertibility of the *G*-matrix for each decoding problem. This was done by selectively choosing encoding matrices at source- and intermediate nodes to determine the effect on the resulting *G*-matrix. For the first experiment, we only used encoding matrices that were invertible at the source nodes. For the second method we attempted only to choose encoding matrices that would, when used to encode data at intermediate nodes, result in invertible output encoding matrices. Finally, we attempted to choose only upper triangular matrices as encoding matrices.

From these experiments, we found that both of the first methods showed improvement in invertibility of the *G*-matrix. The first method showed considerable improvement for RLNC and a slight improvement for MNC while the second showed a substantial improvement for MNC. Most network cases showed little improvement; however, approximately 15% of network cases showed an improvement of 50% or more using Method 2 on MNC. These network cases were comparable to RLNC with Method 1 applied. The final method has never been tested before and produces quite surprising results, since no invertible *G*-matrices were found.

MNC has inherent error-correction abilities, and by limiting the encoding factors we

might influence this capability. We, therefore, tested the influence of network depth and burst errors on networks having only invertible matrices as encoding matrices. The effect of burst errors have not been tested on MNC in previous studies. Among other things, we found that more extensive networks have a lower increase in network error propagation than smaller networks. Further research on the effect of burst errors can still be done using different encoding factors.

Keywords: *Random Linear Network Coding, Matrix Network Coding, Invertibility, Error Correction Coding*

Opsomming

Netwerkkodering kan gebruik word om die maksimum vloei-kapasiteit van 'n netwerk te bereik. Daar bestaan verskeie implementerings van netwerkkodering. Die mees gebruikte netwerkkoderingsmetode is willekeurige lineêre netwerkkodering (RLNC). RLNC is egter baie vatbaar vir netwerkfoute. As gevolg van hierdie nadeel, is matriks netwerkkodering (MNC) in 2011 deur Kim, et al. ontwikkel. MNC verbeter op RLNC, aangesien dit beter fout-korreksie vermoëns het. Aangesien MNC egter matrikse gebruik vir kodering eerder as koëffisiënte, maak dit die stelsel meer ingewikkeld. Dekodering is ook meer ingewikkeld vir MNC, aangesien die *G*-matriks groter is en nie altyd omkeerbaar is, soos wat nodig is vir dekodering, nie.

Ons het op metodes gefokus om die dekodeerbaarheid van MNC te verbeter deur die onkeerbaarheid van die *G*-matriks vir elke dekoderingsprobleem te verbeter. Dit is gedoen deur die enkoderingsmatrikse by bron- en interne nodes selektiewelik te kies om so die invloed op die resulterende *G*-matriks te bepaal. Vir die eerste eksperiment het ons slegs enkoderingsmatrikse gebruik wat by die bronnodes omkeerbaar is, terwyl ons vir die tweede metode probeer het om enkoderingsmatrikse te kies wat sou lei tot omkeerbare matrikse as uitsette vir die internenodes. Laastens het ons slegs bo-driehoekige matrikse gekies as enkoderingsmatrikse.

Uit die eksperimente het ons gevind dat beide die eerste twee metodes 'n verbetering in die omkeerbaarheid van die *G*-matriks getoon het. Die eerste metode het aansienlike verbetering getoon vir RLNC en 'n effense verbetering vir MNC, terwyl die tweede metode 'n aansienlike verbetering vir MNC getoon het. Die meeste netwerke het min verbeter; ongeveer 15% van netwerkgevalle het egter 'n verbetering van 50% of meer getoon met die gebruik van Metode 2 op MNC. Hierdie netwerkgevalle was vergelykbaar met RLNC waarop Metode 1 toegepas is. Die finale metode is nog nie tevore getoets nie en het taamlik verrassende resultate opgelewer, aangesien geen omkeerbare *G*-matrikse gevind is nie.

MNC het inherente foutkorreksievermoëns en deur die enkoderingsfaktore te beperk,

kan ons moontlik die foutkorreksievermoë beïnvloed. Ons het dus die invloed van netwerkdiepte en hoë-digtheid foutlopies op netwerke met slegs omkeerbare matrikse as enkoderingsmatriks getoets. In vorige studies is hoë-digtheid foutlopies nog nie op MNC getoets nie. Ons het onder andere gevind dat meer uitgebreide netwerke 'n laer toename in die verspreiding van netwerkfoute het as kleiner netwerke. Verdere ondersoeke op die effek van hoë-digtheid foutlopies kan nog verder met verskillende enkoderingsfaktore gedoen word.

Sleutelwoorde: *Willekeurige Lineêre netwerkkodering, Matriks netwerkkodering, Omkeerbaarheid, Foutkorreksiekodering*

Contents

List of Figures

List of Tables

List of Acronyms

AWGN Additive White Gaussian Noise

BER Bit Error Rate

BPSK Binary Phase-Shift Keying

BSC Binary Symmetric Channel

CCDF Complimentary Cumulative Distribution Function

EM Electromagnetic

ER Erdős-Rényi

FEC Forward Error Correction

GE Gilbert-Elliot (model)

GF Galois Field

IoT Internet of Things

LNC Linear Network Coding

MNC Matrix Network Coding

NC Network Coding

PNC Physical-layer Network Coding

RLNC Random Linear Network Coding

RS Reed-Solomon

SATNAC Southern Africa Telecommunication Networks and Applications Conference

SNR Signal to Noise Ratio

X-OR Exclusive OR

Chapter 1

Introduction

In this chapter, the importance and relevance of research in Network Coding, and specifically Matrix Network Coding is explained. The chapter contains background on the research area. The research question and objectives are explicitly stipulated, and finally, an overview is given of the chapters in the dissertation.

1.1 Background

In order to keep up with information demands in the modern world, it is crucial to continually improve on the rate that information is sent through networks. Information flow is the rate that information is sent through the network; this should be as high as possible to send data through the network as fast as possible.

The traditional method to send data through a network is by use of routing protocols. When using a routing protocol, data packets are created at the source node and sent into the network. If more than one data packet arrives at any intermediate node at the same time, the packets are sent one after another. In large networks where many

packets can be sent simultaneously, such as is the case with many Internet of Things (IoT), routing protocols are sub-optimal, creating bottlenecks at intermediate nodes [1], [2].

In 2000, Ahlswede, Cai and Li [1] proved that if intermediate network nodes are allowed to combine their incoming packets, and transmit these combinations rather than routing the packets one at a time, the information flow through the network can be increased. This process of combining packets at intermediate network nodes is known as Network Coding (NC).

Since the introduction of network coding, various ways of combining data at intermediate nodes have been proposed. For example, data can be combined on the physical layer by a process known as Physical-layer Network Coding (PNC) using simultaneously arriving Electromagnetic (EM) waves [3]. Another method of combining packets at intermediate nodes is by combining the digital bit streams received at intermediate network nodes.

One of the most influential forms of network coding proposed is Linear Network Coding (LNC) that considers a block of data as a vector over a particular base field [4]. For LNC, linear transformations are applied to any incoming packets before sending them along to the next node. There are various methods of performing LNC. The most widely known method of LNC is Random Linear Network Coding (RLNC), introduced in 2006 [5].

One of the biggest problems associated with RLNC is that bit errors can propagate through the network, corrupting all the data at the destination node [6]. One of the methods to address this problem, Matrix Network Coding (MNC), was proposed in 2011 by Kim *et al.* [7].

Before the research problem can be introduced, some background information on MNC is needed. We will discuss the basics of creating network graphs and how data is traditionally sent through these networks. Then we will discuss the history and development of network coding from concept to linear network coding. More specific

network coding techniques, such as RLNC and MNC, will be discussed in later chapters. Finally, we discuss Galois fields that are used to represent symbols for network coding numerically.

1.1.1 Graph theory

A physical network can be represented mathematically as a graph, where a set of nodes are connected through various edges. The nodes can represent any devices that send or receive data, while edges represent the communication channels between these nodes. The graph is denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of nodes (vertices) connected through a set of channels or edges, $\mathcal{E}$.

A network can be represented as a directed, acyclic graph [1]. The graph is directed since information travels directionally from an input node, i, to an output node, j, through an edge denoted as $(i, j) \in \mathcal{E}$. For any edge $e = (i, j) \in \mathcal{E}$ the head of edge (i, j) is $i = head(e)$ and the tail as $j = tail(e)$.

A graph is considered acyclic if it does not have any directed cycles. An acyclic network, therefore, has a clear order of edges, between intermediate network nodes. An example of this type of graph can be seen in Figure 1.1.

The topology of a network is defined as the interconnection pattern of the nodes in the network [8]. An easy way to represent the graph topology in Figure 1.1 is through an upper triangular connectivity matrix as in Equation (1.1),

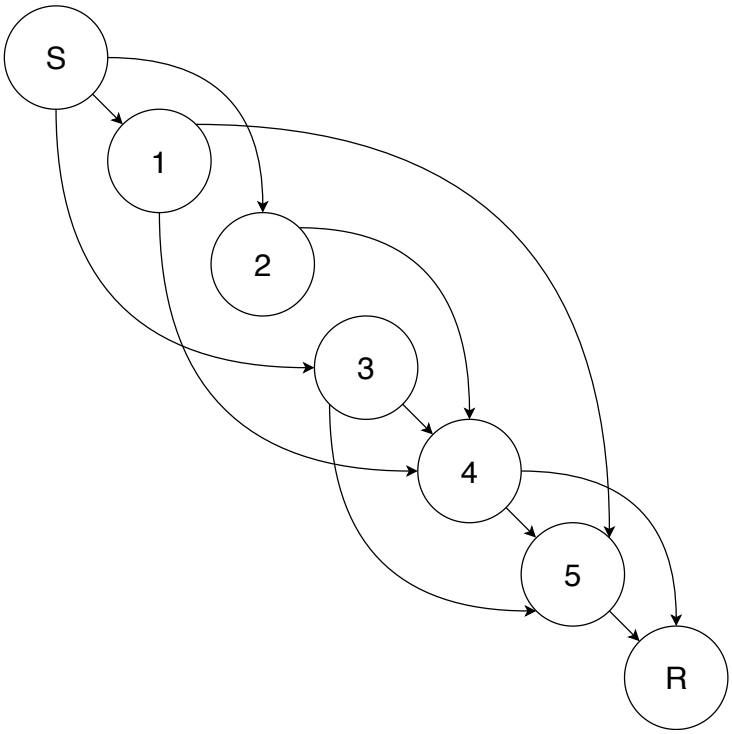

Figure 1.1: Topology of a directed acyclic graph

$$
\begin{array}{c|ccccccc}
 & S & 1 & 2 & 3 & 4 & 5 & R \\
\hline
S & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & & 0 & 0 & 0 & 1 & 1 & 0 \\
2 & & & 0 & 0 & 1 & 0 & 0 \\
3 & & & & 0 & 1 & 1 & 0 \\
4 & & & & & 0 & 1 & 1 \\
5 & & & & & & 0 & 1 \\
R & & & & & & & 0
\end{array}
. \tag{1.1}
$$

In the matrix in Equation (1.1), each row represents a node with the connections it has to nodes ahead of it represented as 1's in the appropriate columns. For example, node 1 is connected to nodes 4 and 5.

A flow network is a network where each node has a capacity and receives flow. As described in [9], information flows through a network as water does through a pipe. Some pipes or channels are smaller than others, and fewer data packets can be sent at

a time. The size of a water pipe is then synonymous with the channel capacity. The throughput, the amount of data that can be transferred in a given amount of time, is dependent on channel capacity.

1.1.2 Min-cut, max-flow theorem

According to [9], the *min-cut, max-flow* theorem is used to determine the highest possible information flow through the network, *max-flow* $= \nu$, using the network topology. The *min-cut* $= \nu$ is the smallest summation of the capacity of the edges that have to be removed to disconnect the source from the destination (sink) node.

The theorem states that the maximum flow that passes from the source to the destination node of the network is equal to the capacity of the minimum cut between those two nodes [9].

We will use the graph in Figure 1.1 condensed to Figure 1.2 as a simple illustration of how the *min-cut, max-flow* theorem can be implemented.

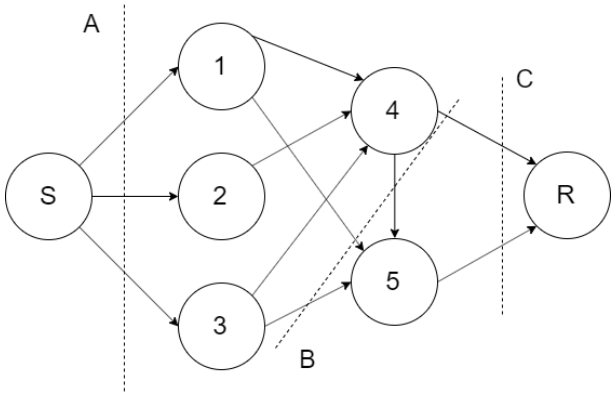

Figure 1.2: Graphical representation of the min-cut max-flow theorem

In Figure 1.2, each channel is assumed to have a capacity of 1. The dotted lines are possible minimum cut lines created for explanation purposes. The max flow for any potential minimum cut line is the flow through that line. In this example, the flow through A is 3, the flow through B is 4 and through C is 2. Of all the possible minimum

cut lines that can be drawn over the figure, line C has the lowest flow and thus, the *min-cut, max-flow* of this network is 2.

1.1.3 The butterfly network

One of the most commonly used network topologies used to explain the benefits of NC is the butterfly network [9]. The network is quite simply a network with one source node, S, two destination nodes, R_1 and R_2 and 4 intermediate nodes (labelled 1 to 4) as seen in Figure 1.3.

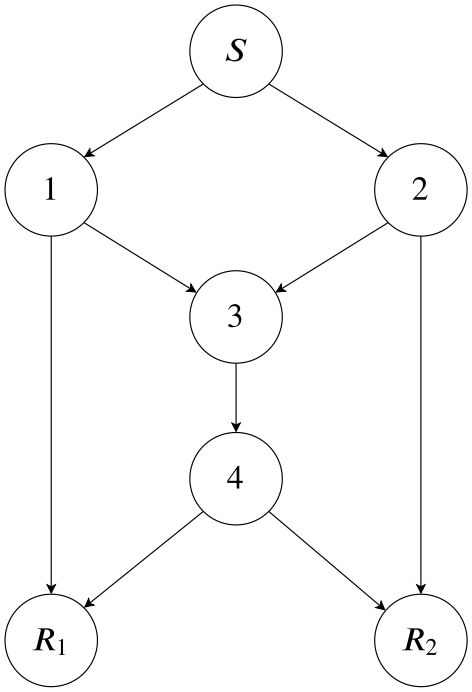

Figure 1.3: Butterfly network [9]

For simplicity, it will be assumed that each edge in the network has a capacity of one, i.e. only one packet can be sent over the network in a single time interval.

1.1.4 Routing

As discussed in [9], over the past few decades almost all computer networks were based on a simple *store-and-forward* or *routing* method. For this method, data is encoded at the source node and sent into the network. At intermediate nodes in the network, data is routed to the destination nodes. If a node has more than one outgoing edge, the data packet is replicated to be sent to all the outgoing edges.

The routing protocol can be applied to the butterfly network in Figure 1.3, where each edge has a capacity of 1. As seen in Figure 1.4, two packets, P_1 and P_2 are sent from the source node, both on different edges. P_1 is sent to node 1 and P_2 is sent to node 2. At nodes 1 and 2 respectively, the packet is replicated and sent out on both the output edges. At node 3, both data packets will be received at the same time. Since the edge capacity is one, only one packet can be sent over the edge between nodes 3 and 4 at a time. For the routing protocol, the packets have to be both stored and sent one after another in two time slots. Whichever packet is received at node 4, at a time, is then also replicated and sent to both the destination nodes.

By sending the packets one at a time between nodes 3 and 4, the packets will use two time slots to be sent over this edge. If we consider the time it takes to send one packet over an edge as one time slot, it will take five time slots for both packets P_1 and P_2 to reach both the destination nodes.

1.1.5 Network coding

In 2000, Ahlswede *et al.* proposed a method now known as Network Coding (NC) that allows the network to achieve bandwidth optimality [1]. NC addresses the problem identified in Section 1.1.4, where the data has to be sent between nodes 3 and 4 over two time slots.

In [1], it is proposed to, rather than send the data one packet at a time, combine the data into a single packet and instead send the combined data packet. This concept is

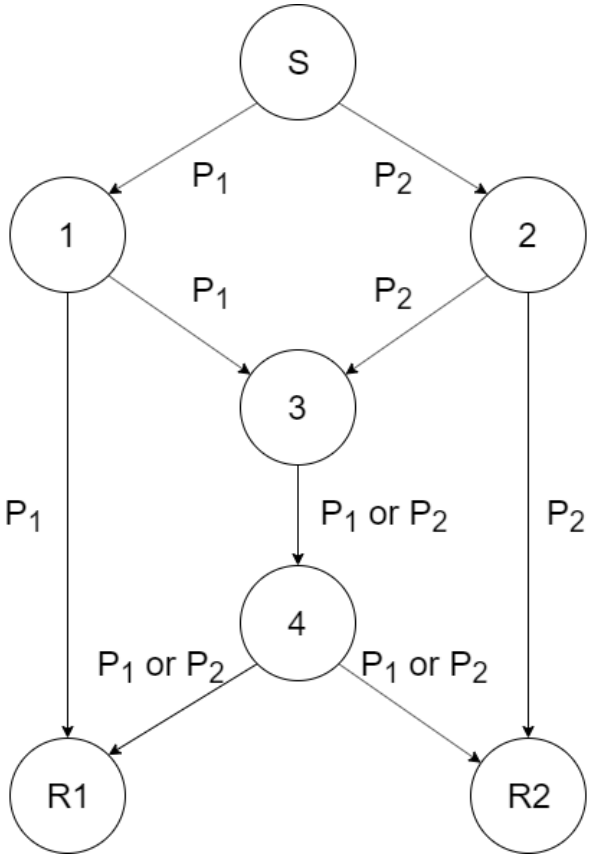

Figure 1.4: Butterfly network with a basic routing protocol

illustrated in Figure 1.5.

In the simplest form of network coding, the data packet can be combined using Exclusive OR (X-OR) operations.

From the example, it follows that the combined packet, P_3 is the same size as either packet P_1 or P_2 due to the X-OR operation. The combination of the packets allows for a faster transmission where only one time slot is used between nodes 3 and 4 rather than the two time slots used for routing.

The example above is quite simplistic; however, in more extensive networks, more information has to be added to the packets to indicate to the destination node which packets were combined to create the received packet. Though this would add more overhead, it also has to be noted that if even more nodes were connected at node 3, it would take proportionally more time intervals to send all the data through the network

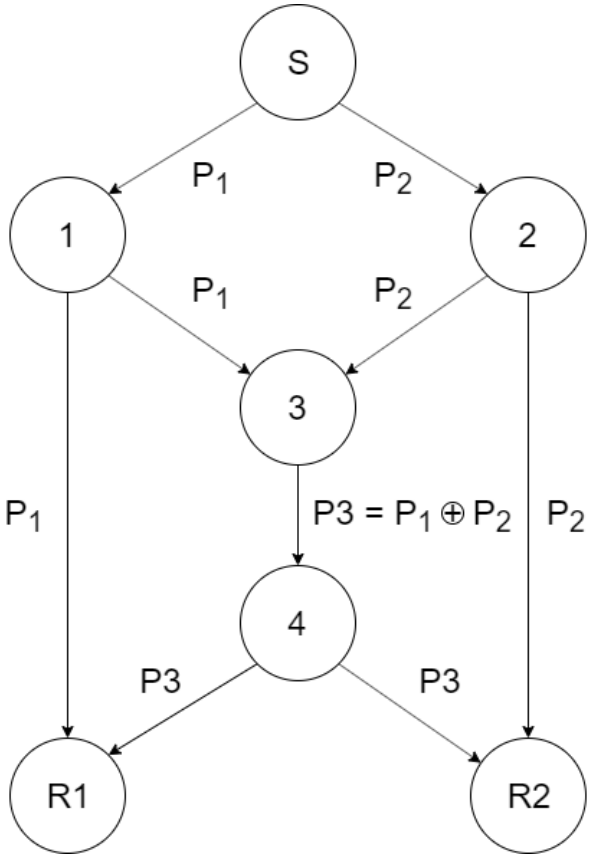

Figure 1.5: Butterfly network on which network coding is implemented

using a routing protocol. By using network coding, the data could still be sent as only one packet, regardless of the number of incoming nodes. This dramatically increases and optimises the throughput of the network as practically determined in [10].

1.1.6 Linear network coding

In 2003, Li, Yeung and Cai [4] defined a sub-field of Network Coding known as Linear Network Coding (LNC). Linear Network Coding is a coding scheme that considers a block of data as a vector over a particular base field. The intermediate nodes can then perform linear transformations of these vectors before passing them along [4]. It was proven in [11] that only linear operations are needed to combine the network packets over a Galois field of symbols. This was done with a deterministic network, a network of which the network topology is known.

A deterministic network, however, is not always the case. Especially for continuously changing networks, the network topology is not always known, and it would take additional time and information to add new nodes to the network. If the network topology is unknown, the network is known as a non-deterministic network topology [12]. Fragouli and Soljanin [13], proposed deterministic algorithms for the intermediate network nodes to code data on non-deterministic networks.

We will discuss two specific types of linear network coding schemes, Random Linear Network Coding (RLNC) and Matrix Network Coding (MNC). These coding schemes will be discussed in-depth in the following chapters.

1.1.7 Galois fields

In NC, a Galois Field (GF) is used to represent a group of bits as an integer symbol [9].

According to [14], a finite field or Galois field is a type of field that contains a finite number of elements. A field is a mathematical set on which multiplication, addition, subtraction and division are defined and follow specific rules. The number of elements in a Galois field is always a positive integer p^m, where, p, is a prime number and the field is defined as F_{p^m} or $GF(p^m)$. F_{2^m} fields are used in information theory since all the numbers in the field can be presented as m bits [14].

From the description, it is important to note that the addition or multiplication of two elements in a Galois field results in another element that is still contained in the same field. As an example the addition and multiplication tables of GF(4) can be seen in eq:intGF [14]:

$$\begin{array}{c|cccc} + & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 2 & 3 \\ 1 & 1 & 0 & 3 & 2 \\ 2 & 2 & 3 & 0 & 1 \\ 3 & 3 & 2 & 1 & 0 \end{array} \qquad \begin{array}{c|cccc} \times & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 2 & 3 \\ 2 & 0 & 2 & 3 & 1 \\ 3 & 0 & 3 & 1 & 2 \end{array} \tag{1.2}$$

1.2 Literature survey

MNC was introduced by Kim et al. in [7]. MNC was created to improve on the network error correction capabilities of RLNC [7]. The paper, however, only served as a conceptual introduction to the routing scheme and minimally small networks were used. It was stated that the Galois Field size adds to the computational complexity of the network. Kim also proposed that encoding factors at intermediate network nodes are chosen such that the output encoding factors of those nodes would be invertible. It was, however, never specified how this could be done.

Due to the improvement in network error correction capabilities, MNC performs well in multihop, multicast scenarios, as shown in [15]. This idea was further expanded on in [16] and [17] by using enhanced MNC techniques such as cache-aided MNC.

Despite the additional network error correction capabilities of MNC, decoding the scheme proves problematic due to the added complexity of matrix encoding factors.

In 2016, Claassen and Helberg [18] did simulations to determine the effect of field size on the invertibility of the G-matrix. It was found that due to the increased size of the encoding matrices, the field size of MNC can be smaller than that of RLNC for equivalent error-correcting capabilities. Similar to Kim, [7], they sugested strategically choosing the encoding matrices so that the encoding matrices at the output of intermediate network nodes would be invertible.

Error correction methods can be added to both MNC and RLNC to aid the decod-

ing process. For RLNC, Forward Error Correction (FEC) can be used to encode each source packet, as discussed in [6]. Since the process of packets propagating through the network, might create more packets than are needed for decoding, these additional packets can be used for error correction even in non-deterministic networks [7]. This is known as intrinsic error correction and is used for MNC.

In 2016, [18] showed that for certain noisy channels, MNC with intrinsic error correction coding over a binary field is more likely to be decoded than RLNC with an inner error correction code. This aligns with [7], that conceptually showed that MNC has a higher error correction capability than RLNC since it provides up to L times minimum distance. Due to the improved error correction capabilities, MNC improves on one of RLNC's most significant weaknesses.

The characteristics of the two Linear Network Coding schemes discussed above can be seen compared in Table 1.1.

Table 1.1: Comparison of two different Network Coding schemes

	RLNC [5]	**MNC** [7]
Coding coefficient	Scalar	Matrix
Deterministic (D) or Non-deterministic (ND)	ND	ND
Field size	F_{2^m}	F_{2^1}
Complexity [7]	$\mathcal{O}(N^3m^2)$	$\mathcal{O}(N^3p^3m^2)$
Error correction research	Yes	Yes

1.3 Research questions

MNC has higher inherent error correction capabilities than RLNC; however, decoding MNC proves difficult due to large encoding factors. The problem that needs to be investigated is if it is possible to choose encoding factors in such a way that MNC is more decodable over small Galois fields.

1.4 Research objectives

Objectives to be addressed when answering the research question is as follows:

- Investigate different ways of improving the invertibility of the G-matrix to improve the decoding probability of MNC for small Galois fields;
- compare the influence of different methods of improving invertibility on the decodability of RLNC and MNC;
- investigate whether invertible output encoding factors at intermediate network nodes result in higher decodability;
- test the effect of network depth on the propagation of errors through a network;
- and test the influence of different intensities of burst errors on the propagation of errors through a network.

1.5 Dissertation overview

The remainder of the dissertation is structured as follows. Chapters 2 and 3 is a literature study on the relevant network coding techniques. In Chapter 2, Random Linear Network Coding will be explained in detail while Matrix Network coding will be discussed in Chapter 3. Chapter 4 concludes the literature study with information on networks, network edges and the relevant error-correcting schemes for MNC and RLNC.

Chapter 5 presents the implementation of RLNC and MNC and how different tests were designed and implemented to answer the research question posed in Section 1.3. The chapter also gives an overview of the verification and validation of the implementation. Chapter 6 presents and discusses the results obtained from the RLNC and MNC coding schemes.

Chapter 7 concludes the dissertation, revisiting the work done in the previous sections and presenting recommendations on future work.

Chapter 2

Random Linear Network Coding

This chapter is part of the literature study in which the network coding technique Random Linear Network Coding is discussed. This chapter contains information on the encoding and decoding of packets using RLNC. An example of how RLNC is also given to explain the concept further.

2.1 Introduction

One of the most widely used and researched forms of network coding is RLNC. This method was proposed in 2006 by Ho *et al.* [5] in which it was proven that the throughput of the network could be achieved by randomly selecting encoding coefficients at each node rather than having them predetermined. Choosing random encoding coefficients allows for non-deterministic networks where the network topology is unknown, or the nodes and connections in the network are dynamically changing.

In this chapter, we discuss RLNC, and specifically how packets are encoded and decoded using this method. The discussion is based on work done primarily by [5] and

elaborated by [19], [7] and [20].

2.2 Encoding

When packets are sent over a network using RLNC, these packets are encoded first at the start node and then at each of the intermediate nodes. An overview of the encoding process can be seen graphically represented in Figure 2.1. The symbols and equations used to encode the data will be listed and discussed in this section.

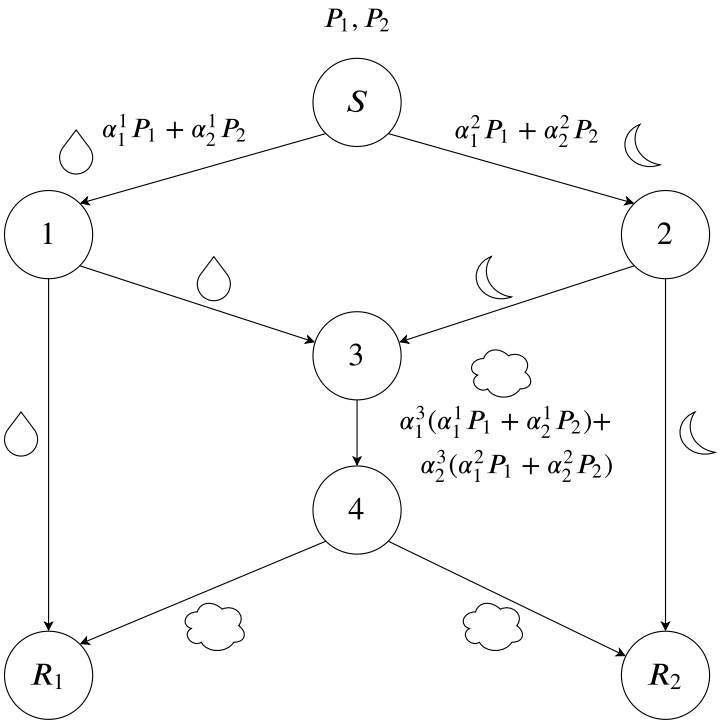

Figure 2.1: Graphical representation of RLNC encoding

2.2.1 Source encoding

For RLNC, the data that will be sent from the source node can be represented as a bitstream. The stream is subdivided into N packets, one for each of the source node's outgoing edges. These packets can be represented as $\mathbf{P} = [P_1, P_2, ..., P_N]$ where each of

the N packets has a length of $L.m$ bits. The bitstream can be defined over a Galois field F_{2^m}, creating L symbols of m bits each. In other words, each data packet is $L.m$ bits or L symbols long.

As mentioned in Section 1.1.2 the network must have a *min-cut* $\geq N$ to avoid capacity constraints.

The output packet P_S^t for each of the outgoing edges, t, of the source node, S, can be calculated as follows:

$$P_S^t = \sum_{i=1}^{N} \alpha_i^t P_i \tag{2.1}$$

In Equation (2.1) local random encoding coefficients, $\mathbf{A}_S^t = [\alpha_1^t, \alpha_2^t, ..., \alpha_N^t]$ are chosen for each outgoing edge. The encoding coefficients are chosen at random over a finite field F_{2^m}. The encoding coefficients at the source node for any outgoing edge, t can be combined into a global encoding vector that can transmitted along with the data on the outgoing edges as seen in Equation (2.2):

$$\mathbf{g}^t = [\alpha_1^t, \alpha_2^t, ..., \alpha_N^t] = [g_1^t, g_2^t, ..., g_N^t] \tag{2.2}$$

with $t = 1 \rightarrow N$ the different outgoing edges.

The outgoing network packets can also be calculated as seen in Equation (2.3), created by combining Equations (2.1) and (2.2):

$$P_S^t = \mathbf{g}^t.\mathbf{P} = \sum_{i=1}^{N} g_i^t P_i \tag{2.3}$$

The encoding vector is sent, along with the newly encoded data, as header information. Figure 2.2 shows the format of the packet that will be sent through the network.

An ID can be added to the start of the packet, along with the header information,

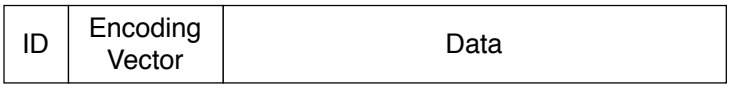

Figure 2.2: Network packet structure

to identify the generation of the packet. The encoding vector will remain the same size throughout the network. From Figure 2.2 it is clear that the header information increases the overhead of each packet.

2.2.2 Encoding at intermediate nodes

The process described above for encoding at source nodes is similar to the encoding process at intermediate network nodes. At an intermediate network node, either one packet or multiple packets can be received at the same time. If only one packet is received, that packet is multi-cast, as is, over all of the node's outgoing edges. If multiple packets are received, they are combined at the intermediate node, and the result of this combination is also multi-cast over all the node's outgoing edges. Note that in both cases only one single packet is transmitted from any intermediate network node.

If, for example, h packets are simultaneously received at an intermediate node, T, these incoming packets are all combined. Note here that h is not necessarily equal to N since the internal structure of a network does not necessarily follow that of the start or end nodes.

In order to combine the h packets, a new set of local encoding coefficients, $\mathbf{A_T} = [\alpha_1, \alpha_2, ..., \alpha_h]$ are randomly chosen from the finite field F_{2^m}. The incoming packets, $P_{T_{in}}$, are then encoded similar to Equation (2.1) as seen in Equation (2.4):

$$P_{T_{out}} = \sum_{j=1}^{h} \alpha_j P_{T_{in}^j} \tag{2.4}$$

From Equation (2.4) it can be seen that only the local encoding coefficients are used to encode the data of the received packets. The headers of the received packets are used to

construct a new header for the outgoing packet, to accompany $P_{T_{out}}$, that combines the global encoding vector g of all h received packets with the h local encoding coefficients $\mathbf{A_T}$.

This combination is done as seen in Equation (2.5) where $\mathbf{g}_{old}$ is an $h \times N$ matrix made up of the global encoding factors $\mathbf{g}^j$ of each of the h incoming packets.

$$\begin{aligned} \mathbf{g}_{new} &= \mathbf{A_T} \cdot \mathbf{g}_{old} \\ g_i &= \sum_{j=1}^{h} \alpha_j g_i^j \end{aligned} \tag{2.5}$$

with $i = 1 \rightarrow N$. This would indicate g_i^j as the i-th element of the header of the j-th incoming packet.

Considering this new header information calculated in Equation (2.5) along with the original source information, we can calculate the output of any intermediate node, $P_{T_{out}}$, as seen in Equation (2.6):

$$P_{T_{out}} = \sum_{i=1}^{N} g_i P_i \tag{2.6}$$

where g_i is the i-th element of the header information and P_i is the i-th of N input packets into the network.

2.3 Decoding

After the data has passed through the entire network, it will eventually reach a destination node where it has to be decoded. Each destination node has M incoming edges. It is important to note that $M \geq N$. In order to decode the source packets, N linearly independent packets are required at each destination node. This allows only for packets that add new information, necessary for decoding purposes.

If, at a destination node, R, N linearly independent network coded packets $P_1, P_2, ..., P_N$ are collected over edges $l = 1 \rightarrow N$, these packets can be calculated as seen in Equation (2.7) using the network input packets and the global encoding coefficients:

$$P_R^l = \sum_{i=1}^{N} g_i^l P_i \tag{2.7}$$

In Equation (2.7), i iterates over the encoding coefficients.

Even more specifically since P_i has L symbols, each output symbol $j = 1 \rightarrow L$ can be calculated as in Equation (2.8):

$$P_R^l(j) = \sum_{i=1}^{N} g_i^l P_i(j) \tag{2.8}$$

The global encoding vectors of each of these packets can be used to form a generator matrix, G. The generator matrix shows the coefficients by which the source packets have been encoded and therefore also how they can be reconstructed. The creation of G can be seen in Equation (2.9):

$$G = [\mathbf{g}^1, \mathbf{g}^2, ..., \mathbf{g}^N] = \begin{bmatrix} \alpha_1^1 & \alpha_1^2 & \cdots & \alpha_1^N \\ \alpha_2^1 & \alpha_2^2 & \cdots & \alpha_2^N \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_N^1 & \alpha_N^2 & \cdots & \alpha_N^N \end{bmatrix} \tag{2.9}$$

Similar to combining the global encoding coefficients into a matrix, each of the j-th symbols of N packets can be combined into a vector, $\tilde{P}_R(j) = (P_R^1(j), P_R^2(j), ..., P_R^N(j))$. Note that $P_R(j)$, that will be used to reconstruct the source packets, is not a single received data packet but rather a combination of symbols from N different incoming packets. The j-th symbols of the N source packets can, similar to the received packets $\tilde{P}_R(j)$, be represented as $\tilde{P}_i(j) = (P_1(j), P_2(j), ..., P_N(j))$.

The j-th elements of the packets received at the destination node can therefore be cal-

culated using $\tilde{P}_i(j)$ and G similar to Equation (2.8) as seen in Equation (2.10):

$$\tilde{P}_R(j) = \tilde{P}(j).G \tag{2.10}$$

for $j = 1 \rightarrow L$. In Equation (2.10) we see a set of linear equations over a finite field F_{2^m}. If G is invertible, we can solve the equation for the source packets as seen in Equation (2.11):

$$\tilde{P}(j) = \tilde{P}_R(j).G^{-1} \tag{2.11}$$

for $j = 1 \rightarrow L$ denoting the j-th element in the source packet.

2.4 Example

Consider an example, where data 110|101|010|100 have to be sent through the butterfly network, examples of which can be seen in Figure 2.1. The data has to reach both destination nodes, R_1 and R_2. If we apply RLNC as discussed in this section, the data is divided into two packets and represented as Galois field numbers. For this example, the Galois field is chosen as F_{2^3} or $GF(8)$. This results in the packets at the source being, $P_1 = [6, 5]$ and $P_2 = [2, 4]$.

For RLNC, various addition and multiplication operations are required over the Galois field. The addition (left) and multiplication tables (right) for $GF(8)$ can be seen in Equation (2.12) to be used as reference when calculating new packets and header information.

$+$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

$\times$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

(2.12)

At the source node, the packets are encoded for each outgoing edge. For each outgoing edge, two encoding coefficients are chosen at random. The two encoding coefficients are chosen due to the two outgoing edges, $N = 2$. For the first edge, the outgoing coefficients are chosen as $g^1 = [\alpha_1^1, \alpha_2^1] = [5, 2]$ as in Equation (2.2). The output of the first edge can be calculated using Equation (2.1) as seen in Equation (2.13):

$$\begin{aligned} P_S^1 &= \sum_{i=1}^{2} \alpha_i^1 P_i \\ &= \alpha_1^1 P_1 + \alpha_2^1 P_2 \\ &= 5 \cdot [6,5] + 2 \cdot [2,4] \\ &= [3,7] + [4,3] \\ &= [7,4] \end{aligned} \tag{2.13}$$

If the same process is followed for the second outgoing edge, with $g^2 = [\alpha_1^2, \alpha_2^2] = [4, 3]$ the output of the second edge will be $P_S^2 = [3, 5]$.

The resulting packets that will be sent from the source node can be seen in Figure 2.3.

At nodes 1 and 2, respectively, only one packet is received. This packet is, therefore, forwarded as is on all the outgoing edges of the particular nodes.

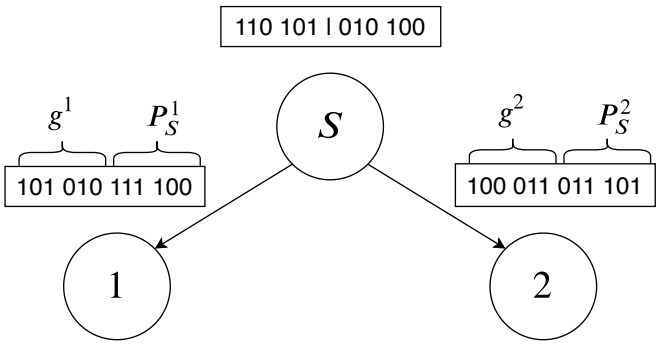

Figure 2.3: Example of packet encoding at the source node using RLNC

Two packets arrive from nodes 1 and 2 at node 3 simultaneously. These packets are combined to create a new output packet. Since there are two incoming edges at this intermediate node, $h = 2$. Since $h = 2$, two new local encoding coefficients are randomly chosen as, $\mathbf{A_T} = [\alpha_1, \alpha_2] = [5, 4]$. Using Equation (2.4) we can calculate the output packet, $P_{T_{out}}$, of the intermediate node as seen in Equation (2.14)

$$\begin{aligned} P_{T_{out}} &= \sum_{j=1}^{2} \alpha_j P_{T_{in}^j} \\ &= \alpha_1 P_{T_{in}^1} + \alpha_1 P_{T_{in}^1} \\ &= 5 \cdot [7, 4] + 4 \cdot [3, 5] \\ &= [6, 2] + [7, 2] \\ &= [1, 0] \end{aligned} \tag{2.14}$$

For the header information at node 4, the information is calculated according to Equation (2.5) as seen below in Equation (2.15):

$$
\begin{aligned}
g_{new} &= \mathbf{A_T} \cdot g_{old} \\
&= [5,4] \cdot \begin{bmatrix} 5 & 2 \\ 4 & 3 \end{bmatrix} \\
&= [5 \cdot 5 + 4 \cdot 4, 5 \cdot 2 + 4 \cdot 3] \\
&= [7 + 6, 1 + 7] \\
&= [1,6]
\end{aligned}
\tag{2.15}
$$

The input and output packets created at node 3 can be seen in Figure 2.4.

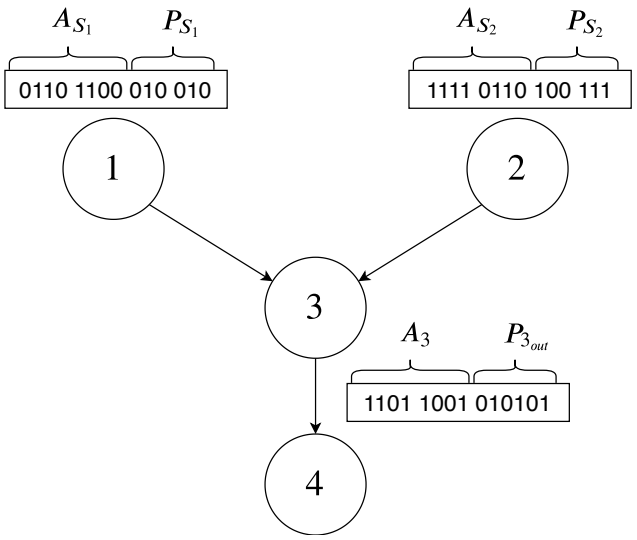

Figure 2.4: Example of packet encoding at the intermediate node using RLNC

The destination node R_1, receives two encoded packets, one from node 1 and one from node 4. Similarly R_2 receives packets from nodes 2 and 4, as seen in Figure 2.5

At node R_1, the header information from the two received packets is used to create the G-matrix. The G-matrix is created according to Equation (2.9) with $N = 2$ as seen in Equation (2.16)

$$
G = [\mathbf{g}^1, \mathbf{g}^2] = \begin{bmatrix} \alpha_1^1 & \alpha_1^2 \\ \alpha_2^1 & \alpha_2^2 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 2 & 6 \end{bmatrix} \tag{2.16}
$$

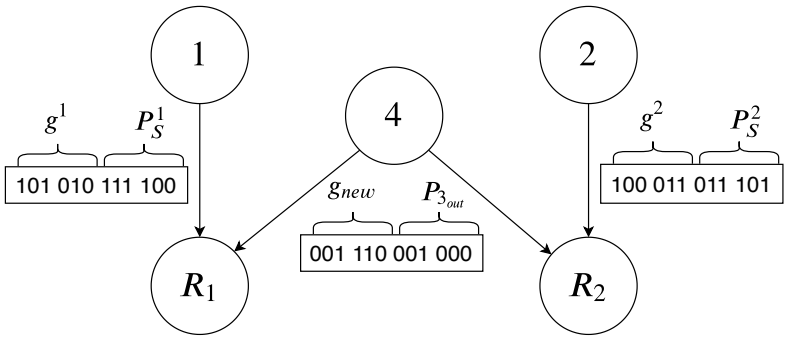

Figure 2.5: Example of RLNC encoded packets received at the destination nodes of a butterfly network

The inverse of the G-matrix which is required for decoding the input can be calculated as seen in Equation (2.17)

$$G^{-1} = \begin{bmatrix} 6 & 1 \\ 2 & 5 \end{bmatrix} \tag{2.17}$$

For each data symbol in the data packets, $j = 1 \rightarrow L$, the source packets can be calculated using Equation (2.11). For destination node R_1, the received data packets are $[7,4]$ and $[1,0]$. Since both the data packets are two Galois field symbols long, $L = 2$. The packets are first combined by symbol to be $\tilde{P}_R(1) = [7,1]$ and $\tilde{P}_R(2) = [4,0]$.

Using $\tilde{P}_R$ and G^{-1} we can calculate the incoming packets. In Equation (2.18), for example, we determine the first symbols of the input packets, $\tilde{P}(1)$, using the first symbols of the received packets, $\tilde{P}_R(1)$:

$$\begin{aligned} \tilde{P}(1) &= \tilde{P}_R(1) \cdot G^{-1} \\ &= [7,1] \cdot \begin{bmatrix} 6 & 1 \\ 2 & 5 \end{bmatrix} \\ &= [7 \cdot 6 + 1 \cdot 2, 7 \cdot 1 + 1 \cdot 5] \\ &= [4 + 2, 7 + 5] \\ &= [6,2] \end{aligned} \tag{2.18}$$

From Equation (2.18) it is clear that the first symbol in the result is the first symbol of

the first data packet and the second symbol is the first symbol from the second data packet before encoding. Similarly with $\tilde{P}_R(2)$, $\tilde{P}(2)$ can be calculated as $[5, 4]$.

The input of R_2 can be used similarly to recalculate the input data of the network and will yield the same results as for R_1.

2.5 Conclusion

RLNC is a popular network coding technique that can be used in networks with unknown or dynamically changing topologies. We discussed how this method is applied by encoding packets at both the source node and intermediate network nodes. This was demonstrated through a small example. Packets are encoded using encoding coefficients that are randomly selected. The randomly selected coefficients can also be selected more precisely according to some parameters.

One prominent drawback of RLNC is the possibility that errors can be catastrophically propagated through the network. The propagation can be mitigated using a FEC technique as will be discussed in Section 4.4.

Another drawback of RLNC is that it requires an invertible generator matrix at the destination node in order to be decoded. One method proposed to improve on this is the more recently proposed MNC.

Chapter 3

Matrix Network Coding

The chapter on MNC is also part of the literature study in which the network coding technique Matrix Network Coding is discussed. This chapter contains information on the encoding and decoding of packets using MNC. An example of the process is also given. The chapter concludes with a comparison of MNC and RLNC.

3.1 Introduction

MNC was introduced in 2011 by Kim *et al.* as a non-deterministic coding scheme that improves on the network error correction capabilities of RLNC [7]. The implementation of MNC and RLNC is functionally the same, the significant difference between the methods is that instead of the encoding coefficients used for RLNC, MNC uses matrices as encoding coefficients. These encoding coefficients are created by randomly choosing the symbols over a finite field, F_{2^m}. Since the size of the encoding matrix can be varied, it is sufficient to limit the field to only binary numbers, F_2 as will become apparent through the discussion.

We follow the explanation of MNC as discussed in [7] and [20] with slight adjustments to cohere to the logical explanation given in Section 3.4. In this chapter, we will discuss how encoding and basic decoding is done to implement MNC.

3.2 Encoding

The main difference between MNC and RLNC is the size of the encoding coefficient, as already mentioned. These encoding coefficients for MNC are chosen as $p \times p$ matrices. To encode the source information with these matrices, the source packets also have to be presented as matrices.

3.2.1 Defining data packets

Similar to RLNC, the source node has N packets, $P_1, P_2, ..., P_N$ defined over a finite field F_{2^m} corresponding to the N output edges of the source node. These packets have a fixed length of L symbols. For MNC, these data packets are represented as p-dimensional vectors of q symbols, over F_{2^m}. q can be determined from the length of the data packet, as $L = pq$ symbols over a finite field F_{2^m}.

The data matrices can be constructed by representing the input data, P_i, as p, q-dimensional vectors using Equation (3.1):

$$P_i = (\mathbf{P}_i(1), \mathbf{P}_i(2), ..., \mathbf{P}_i(p)) \tag{3.1}$$

with all of the q-dimensional vectors as seen in Equation (3.2):

$$\mathbf{P}_i(j) = (P_i((j-1)q+1), P_i((j-1)q+2), ..., P_i(jq)) \tag{3.2}$$

for all $j = 1 \rightarrow p$. In Equations (3.1) and (3.2), P_i represents all the different input

packets to the source node for $i = 1 \rightarrow N$.

This rearrangement of the symbols of the source data results in a $q \times p$ data matrix, where data is grouped into p columns of q symbols each. This is done to all the data packets from the start node, intermediate nodes and the destination node. The data is always represented in this format when it is captured.

As an example, consider a source packet P_1 that has 6 symbols $[P_1(1), P_1(2), ..., P_1(6)]$ over F_{2^m}. If we use an MNC scheme with $p = 2$, we know $q = 3$ since $L = pq$ and L is the number of symbols in the packet, which in this case is 6. We can, therefore, represent the data of P_1 as 2 columns of 3 symbols each as seen in Equation (3.3):

$$P_1 = \begin{bmatrix} P_1(1) & P_1(2) \\ P_1(3) & P_1(4) \\ P_1(5) & P_1(6) \end{bmatrix} \tag{3.3}$$

3.2.2 Source encoding

Suppose that at the source node, N data packets have to be sent. These N packets, $P_1, P_2, ..., P_N$, are all represented as $q \times p$ matrices as discussed in Section 3.2.1 in Equations (3.1) and (3.2).

For these N packets, N encoding matrices, $A_S^1, A_S^2, ...A_S^N$, consisting of $p \times p$ symbols, are constructed for each outgoing edge at the source node. Each of the elements is chosen from the finite field F_2^m. Since N encoding matrices are created per output edge, a total of N^2 independent encoding matrices are created at the source node.

Encoding at the source node is similar to that of RLNC as seen in Equation (2.1) where the coefficient α is now expanded from a single symbol to a matrix of symbols A_S. Encoding can take place for each of the q rows as seen in Equation (3.4):

$$\mathbf{P}_S(j) = \sum_{i=1}^{N} \mathbf{P}_i(j) \cdot A_S^i \tag{3.4}$$

where $j = 1 \rightarrow q$. More explicitly,

$$\mathbf{P}_S(j) = \mathbf{P}_1(j)A_S^1 + \mathbf{P}_2(j)A_S^2 + \cdots + \mathbf{P}_N(j)A_S^N \tag{3.5}$$

where $j = 1 \rightarrow q$. Note that each data packet $P_i(j)$ contains p symbols which allows the matrix multiplication required in Equation (3.5). Each row, j of the source data packets is therefore encoded separately and then added together to create the new source data matrix as seen in Equation (3.6)

$$P_S = (\mathbf{P}_S(1); \mathbf{P}_S(2); \cdots ; \mathbf{P}_S(q)) \tag{3.6}$$

3.2.3 Header information representation

The N encoding coefficients are each represented as $p \times p$ matrices. Each of these encoding coefficients consist of p^2 randomly selected symbols α_i where $j = 1 \rightarrow p^2$ that were chosen over a finite field F_{2^m}. These matrices have to be sent as header information along with the newly encoded data. To accomplish this, the matrix, as with the data information in Section 3.2.1, can be written as a one-dimensional vector for transmission.

If $p = 2$, a matrix encoding coefficient, A_i can be written as seen in Equation (3.7):

$$A_i = \begin{bmatrix} \alpha_1 & \alpha_3 \\ \alpha_2 & \alpha_4 \end{bmatrix} \rightarrow \beta_i = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{bmatrix} \tag{3.7}$$

In Equation (3.7), β_i represents the new formation of the single encoding coefficient. In order to construct the header of the outgoing packet, each of the N encoding coeffi-

cients have to be written in the same form of β_i. The global encoding vector that is sent as the header information can then be constructed as a vector consisting of vectors as seen in Equation (3.8):

$$\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_N] \tag{3.8}$$

$\boldsymbol{\beta}$ in Equation (3.8) is Np^2 symbols long since each β_i is p^2 symbols long.

It is important to note that only at the source node can the local encoding coefficients be directly used to construct the header of the packet. The encoding coefficients of the intermediate network codes will be discussed in Section 3.2.4 below.

3.2.4 Intermediate node encoding

The data received at intermediate nodes are processed in the same manner as for RLNC discussed in Section 2.2.2. If only one data packet is received, it is broadcast, as is, over all the outgoing edges. If, however, more than one data packet is received simultaneously, these packets are combined, and the combined packet is broadcast over all outgoing edges.

Say h data packets are received at an intermediate node T, a new set of encoding matrices $A_1, A_2, ..., A_h$ are generated, with elements randomly chosen from the finite field F_{2^m}. The incoming packets, $P_{T_{in}}$, are encoded as seen in Equation (3.9):

$$\mathbf{P}_{T_{out}}(j) = \sum_{r=1}^{h} \mathbf{P}^r_{T_{in}}(j) A_r \tag{3.9}$$

for $j = 1 \rightarrow q$. As with RLNC, only local encoding coefficients are used to encode the data packets at intermediate network nodes.

The local encoding coefficients can be written as described in Equation (3.8) and the global encoding vectors can be updated as seen in Equation (3.10):

$$\boldsymbol{\beta}_{(new)} = \boldsymbol{\beta}_{(old)} \cdot \mathbf{A_T}$$
$$\beta_i = \sum_{r=1}^{h} \beta_i^r \cdot A_r \tag{3.10}$$

for $i = 1 \rightarrow N$.

From the differences mentioned between RLNC and MNC thus far, it should be noted that MNC reduces to RLNC if $p = 1$. The encoding matrices will reduce to encoding coefficients, and L will be left as $L = q$. For this reason, in this paper, the term MNC implies $p \geq 2$.

3.3 Decoding

At the destination node, h different matrix combined packets will be received that are decoded using the matrix encoding coefficients found in each packet header. The linearly independent encoding coefficients are combined into a matrix known as the G-matrix seen in Equation (3.11):

$$G = \begin{bmatrix} \boldsymbol{\beta}^1 & \boldsymbol{\beta}^2 & \cdots & \boldsymbol{\beta}^h \end{bmatrix} = \begin{bmatrix} A_1^1 & A_1^2 & \cdots & A_1^h \\ A_2^1 & A_2^2 & \cdots & A_2^h \\ \vdots & \vdots & \ddots & \vdots \\ A_N^1 & A_N^2 & \cdots & A_N^h \end{bmatrix} \tag{3.11}$$

This matrix is similar to the one defined for RLNC in Equation (2.9) except that each factor A_i^j in the matrix is, in fact, itself a $p \times p$ matrix. This results in the G-matrix being a $N \cdot p \times h \cdot p$ matrix. With standard MNC decoding, only N of the h incoming packets will be used. This leaves the G-matrix to be $N \cdot p \times N \cdot p$. The remaining packets can be used for error-correcting purposes, as will be discussed in Section 4.5.

The N data packets are combined into j packets by taking the first row of each data packet and combining this into the output packet, $\tilde{\mathbf{P}}_o$ as seen in Equation (3.12)

$$\tilde{\mathbf{P}}_o(j) = (\mathbf{P}_o^1(j), \mathbf{P}_o^2(j), \cdots, \mathbf{P}_o^N(j)) \tag{3.12}$$

for $j = 1 \rightarrow q$. Using $\tilde{\mathbf{P}}_o$ and the G-matrix we can determine the original source packets as seen in Equation (3.13) similar to Equation (2.11).

$$\tilde{\mathbf{P}}(j) = \tilde{\mathbf{P}}_o(j).G^{-1} \tag{3.13}$$

for $j = 1 \rightarrow q$. In Equation (3.13), $\tilde{\mathbf{P}}(j)$ represents the j-th rows of all the input packets as seen in Equation (3.14):

$$\tilde{\mathbf{P}}(j) = (\mathbf{P}_1(j), \mathbf{P}_2(j), \cdots, \mathbf{P}_N(j)) \tag{3.14}$$

for $j \rightarrow q$. Decoding is done with the assumption that no errors occurred during transmission.

3.4 Analytical basis

Here we will explain the analytical basis for why packets are encoded at the source node and why they can then be decoded at the end node. This analytical explanation is for a simple four-node network as seen in Figure 3.1.

For this explanation, no internal network coding is needed since only the basics of encoding and decoding are explained. For the example, the source has two output edges, $N = 2$ and we choose $p = 2$.

At the source node, data packets P_1 and P_2 have to be encoded. For the first outgoing edge, these packets are encoded with encoding matrices A and C to produce the packet P_{O1}. For the second outgoing edge, the packets are encoded with the encoding matrices B and D to produce the packet P_{O2}.

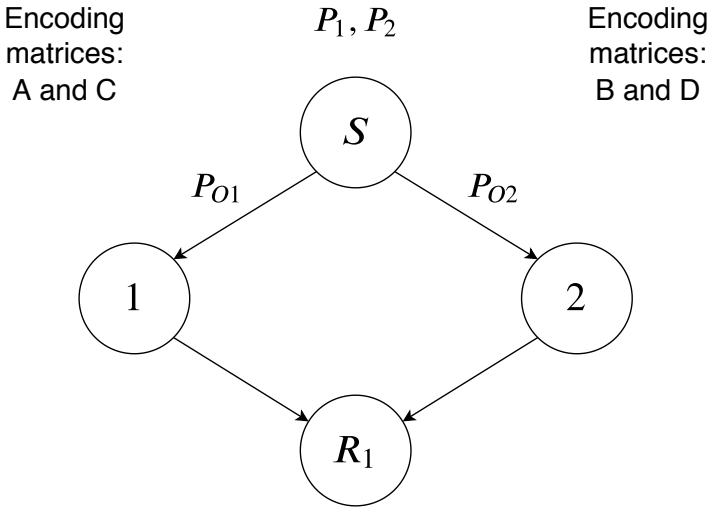

Figure 3.1: Network of 4 nodes aiding in the analytical description of MNC source encoding

These calculations can be seen in Equation (3.15):

$$\begin{aligned} P_{O1} &= P_1 A + P_2 C \\ P_{O2} &= P_1 B + P_2 D \end{aligned} \tag{3.15}$$

Our goal is to rewrite one of the input packets, P_2, in terms of the output packets to compare it with the decoding method.

We can rewrite the input packet P_1 in terms of the other packets,

$$P_1 = (P_{O1} - P_2 C)A^{-1} \tag{3.16}$$

Replacing Equation (3.16) into the equation for P_{O2} yields,

$$\begin{aligned} P_{O2} &= (P_{O1} - P_2 C)A^{-1}B + P_2 D \\ &= P_{O1}A^{-1}B + P_2(D - CA^{-1}B) \end{aligned} \tag{3.17}$$

From Equation (3.17), P_2 can be determined as in Equation (3.18).

$$P_2 = (P_{O2} - P_{O1}A^{-1}B)(D - CA^{-1}B)^{-1} \tag{3.18}$$

We can now compare this packet P_2 to the method in which P_{O1} and P_{O2} are used to reattain P_2. At the receiver, R_1, the encoding factors are combined into a G-matrix as seen in Equation (3.19):

$$G = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{3.19}$$

the G-matrix can be inverted as seen in Equation (3.20):

$$G^{-1} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \left[\begin{array}{c|c} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{array}\right] \tag{3.20}$$

G^{-1} is required according to Equation (3.13) to decode the input packets. Since P_2 can be determined by multiplying P_O with the second column of G^{-1}, we can determine P_2 as seen in Equation (3.21):

$$P_2 = -P_{O1}A^{-1}B(D - CA^{-1}B)^{-1} + P_{O2}(D - CA^{-1}B)^{-1} \tag{3.21}$$

It is clear that Equations (3.18) and (3.21) are equal. This shows that the data that was encoded at the source node using encoding coefficients can be decoded using the G-matrix at the destination node.

3.5 Example

The example for MNC will follow a similar structure to the RLNC example in Section 2.4 and can be compared side by side to understand the similarities and differences

between the methods. Consider an example, where the same data packet as for RLNC, 110 101| 010 100, has to be sent through the butterfly network.

The data has to reach both destination nodes, R_1 and R_2. If we apply MNC as discussed in this chapter, the data is divided into two packets and represented as Galois field numbers. For the purpose of this example the Galois field is chosen as F_{2^1} or $GF(2)$ which is normally the case with MNC. The two packets, of size $L = 6$, are then changed to matrices as seen in Equation (3.2). For our example, $p = 2$ and $q = L/p = 3$. Therefore, the data is combined into matrices with 3 rows and 2 columns each. This results in the packets at the source being,

$$P_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad P_2 = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Calculations will be done over $GF2$ of which the addition (left) and multiplication tables (right) can be seen in Equation (3.22) to be used as a reference when calculating new packets and header information.

XOR

$+$	0	1
0	0	1
1	1	0

AND

$\times$	0	1
0	0	0
1	0	1

(3.22)

At the source node, the packets are encoded for each outgoing edge. For each outgoing edge, two source encoding matrices, of size 2×2, are chosen at random.

$$A^1_{S_1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad A^2_{S_1} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

$$A^1_{S_2} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \text{and} \qquad A^2_{S_2} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The output of the first edge can be calculated using Equation (3.4), as seen in Equation (3.23):

$$\begin{aligned} \mathbf{P}_{S_1}(j) &= \sum_{i=1}^{2} \mathbf{P}_i(j) A^i_{S_1} \\ &= \mathbf{P}_1(j) A^1_{S_1} + \mathbf{P}_2(j) A^2_{S_1} \\ \mathbf{P}_{S_1}(1) &= \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \end{bmatrix} \end{aligned} \tag{3.23}$$

Equation (3.23) only shows the first row of the full source packet, P_{S_1}. The full source packets for the first edge, P_{S_1} and for the second edge P_{S_2} can be seen in Equation (3.24) below:

$$P_{S_1} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \text{ and } P_{S_2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \tag{3.24}$$

The header information will be presented as in Equation (3.8). The resulting packets that will be sent from the source node can be seen in Figure 3.2.

Two packets arrive from nodes 1 and 2 at node 3 simultaneously. These packets are combined to create a new output packet. Since there are two incoming edges at this intermediate node, $h = 2$. Since $h = 2$, two new local encoding matrices are randomly chosen as seen in Equation (3.25).

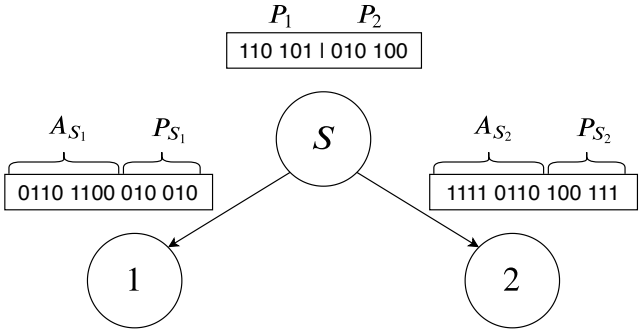

Figure 3.2: Example of packet encoding at the source node using MNC

$$\mathbf{A_T} = \{A_1, A_2\} = \left\{ \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right\} \tag{3.25}$$

Using Equation (3.9) we can calculate the output packet of the intermediate node, $\mathbf{P}_{T_{out}}$ for each row. In Equation (3.26), we can see this for the first row or $j = 1$.

$$\begin{aligned} \mathbf{P}_{T_{out}}(j) &= \sum_{r=1}^{2} \mathbf{P}^{r}_{T_{in}}(j) A_r \\ &= \mathbf{P}^{1}_{T_{in}}(1) A_1 + \mathbf{P}^{2}_{T_{in}}(1) A_2 \\ &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \end{bmatrix} \end{aligned} \tag{3.26}$$

The total output of the intermediate node can be similarly calculated as seen in Equation (3.27):

$$P_{T_{out}} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \tag{3.27}$$

For the header information at node 4, the information is calculated according to Equation (3.10) as seen below in Equation (3.28) for $i = 1$, $\beta_1^1 = A_{S_1}^1$ and $\beta_1^2 = A_{S_2}^1$:

$$\begin{aligned}
\beta_i &= \sum_{r=1}^{2} \beta_i^r \cdot A_r \\
\beta_1 &= \beta_1^1 A_1 + \beta_1^2 A_2 \\
&= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.28}$$

The second encoding matrix at node 4 can be calculated in a similar manner as seen in Equation (3.29):

$$\beta_2 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \tag{3.29}$$

The input and output packets created at node 3 can be seen in Figure 3.3.

The destination node R_1, receives two encoded packets, one from node 1 and one from node 4. Similarly R_2 receives packets from nodes 2 and 4, as seen in Figure 3.4.

At node R_1, the header information from the two received packets is used to create the G-matrix. The G-matrix is created according to Equation (3.11) with $N = 2$ and $h = 2$ as seen in Equation (3.30).

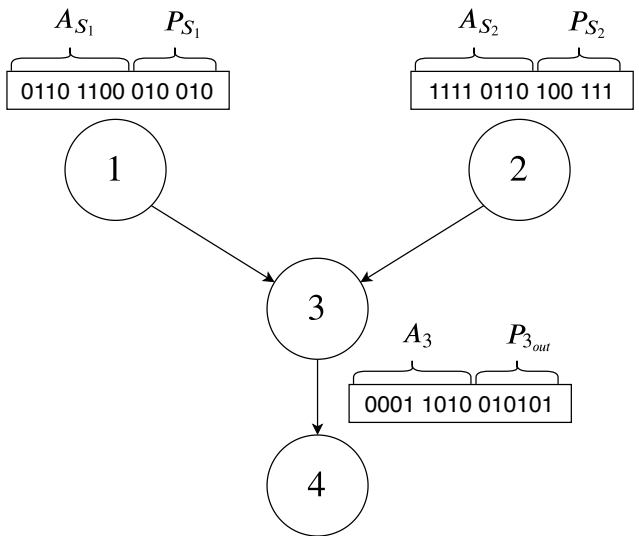

Figure 3.3: Example of packet encoding at the intermediate node using MNC

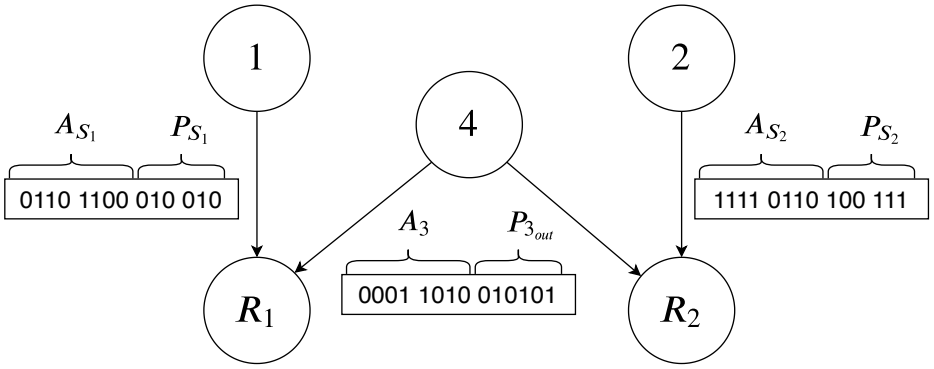

Figure 3.4: Example of MNC encoded packets received at destination nodes of a butterfly network

$$G = \begin{bmatrix} \beta^1 & \beta^2 \end{bmatrix} = \begin{bmatrix} A_1^1 & A_1^2 \\ A_2^1 & A_2^2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.30}$$

The inverse of the *G*-matrix which is required for decoding the input can be calculated as in Equation (3.31):

$$G^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.31}$$

or each row in the data packets, $j = 1 \rightarrow q$, $\tilde{\mathbf{P}}_R(j)$ can be created. $\tilde{P}_R$ is created by placing the two received data packets next to one another as $\tilde{P}_R = \begin{bmatrix} P_{S_1} & P_{T_{out}} \end{bmatrix}$. The input packets, $\tilde{P}$ can be calculated as seen in Equation (3.32):

$$\begin{aligned} \tilde{P} &= \tilde{P}_R \cdot G^{-1} \\ &= \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \\ &= \left[\begin{array}{cc|cc} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right] \end{aligned} \tag{3.32}$$

The decoded packets from Equation (3.32) are the input packets, $\tilde{P} = \left[\begin{array}{c|c} P_1 & P_2 \end{array}\right]$. The same results can be seen when decoding at R_2.

3.6 Comparison of RLNC and MNC

RLNC and MNC are quite similar as can be seen through the cross-references throughout this chapter. MNC can be seen as an expansion to RLNC since the most considerable difference between the two is that for RLNC encoding coefficients are comprised of one symbol, while for MNC, encoding matrices of size $p \times p$ are used.

For both RLNC and MNC, the input packets can only be recovered if the G-matrix is

invertible.

The most significant drawback of RLNC is that errors can be propagated through the network, causing catastrophic errors at the destination node. This is mitigated using MNC since the additional information inherent to MNC allows for intrinsic error correction [7]. Additional packets received at the destination node can also be used for error correction. This is the primary consideration for using MNC.

MNC also comes with drawbacks, the most important of which can be seen from Equation (3.13). The complexity of decoding can be troublesome with larger G-matrices. According to [7], if the header overhead of MNC and RLNC is kept the same, the complexity of MNC is estimated as $\mathcal{O}(N^3p^3m^2)$ while RLNC in comparison has a complexity of $\mathcal{O}(N^3m^2)$ when N source packets are combined, at each node, $p \times p$ matrix chosen over a finite field F_{2^m}.

The field size for MNC can be lower than for RLNC since MNC has bigger encoding matrices [7]. Kim noted that if only a binary field, m_1, is used for MNC and $m = m_1p^2$ for RLNC, the complexity of MNC would be $\mathcal{O}(N^3p^3{m_1}^2)$ compared to $\mathcal{O}(N^3p^4{m_1}^2)$ for RLNC [7]. Therefore, the complexity of RLNC would be slightly higher if a binary field size is used for MNC and the RLNC field is p^2 times that of MNC.

The final drawback of both RLNC and MNC is that the G-matrix must be invertible, which is often not the case for small field sizes.

3.7 Conclusion

MNC is very similar to RLNC but improves on its error correction ability. We discussed how MNC is applied by encoding packets at both the source node and intermediate network nodes and then decoding the packets at the destination node. This was demonstrated practically through a small example. Contrary to RLNC, packets are encoded using encoding matrices that are randomly selected rather than coefficients.

The randomly selected matrices can also be selected more precisely according to some parameters.

MNC can also be used in unknown topologies that will be discussed in the following chapter. Additional error correction techniques for MNC will also be discussed in Section 4.5.

Chapter 4

Networks, Errors and Error Correction

This chapter forms the final part of the literature study. It discusses Erdős Rényi graphs, Binary Symmetric Channels, the Gilbert Elliot channel model, and the error correction techniques, Forward Error Correction and Coset Decoding.

4.1 Introduction

In order to do any testing on the network coding methods of MNC and RLNC, these techniques have to be applied to real networks. Real networks are often affected by errors. We will first look at the simple Erdős Rényi graph as a channel model to represent the networks.

4.2 Erdős Rényi graphs

The original paper on MNC [7] only included a simple graph consisting of 4 nodes for testing. We will expand on this simple graph, as done in [15] and [20], by also

considering the Erdős-Rényi (ER) graph.

An ER-graph created by introducing random edges, E, between the nodes, V, of a graph. The edges are added independently of one another, all with the same probability δ to create the graph $G = \mathbf{ER}(V, \delta)$ [21].

To create the entire network for simulations, including the source and the destination node, we will generate the internal network nodes using an ER-graph model and add a source and destination node to the graph. The source node is connected through N edges to N different nodes in the internal network in order to satisfy the $min - cut$, $max - flow$ theorem. The destination node is connected to M different nodes of the network. M depends on the decoding scheme implemented, but it is presumed $M \geq N$, again to satisfy the $min - cut$, $max - flow$ theorem. An example of a graph can be seen in Figure 4.1:

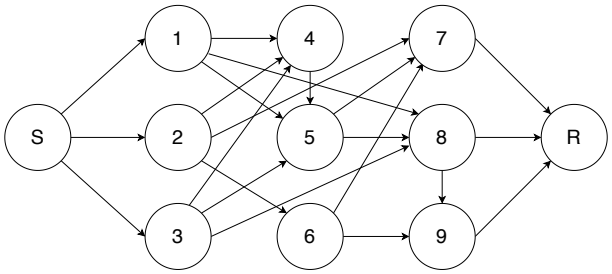

Figure 4.1: Example of an Erdős Rényi graph with 9 intermediate network nodes, an added source node (S), destination node (R) and $N = M = 3$

A higher connectivity probability leads to more densely connected graphs. If graphs are too sparsely connected, the $min - cut$, and, thus, the $max - flow$ of the network is lowered. The probability of connectivity in the network has to be chosen accordingly, and a network that does not satisfy $min - cut \geq N$ will not be evaluated.

4.2.1 Assumptions on graph connectivity

Some assumptions were made to simplify the calculations and to increase the probability of network coding:

1. The capacity of all network edges is 1.
2. We will also not take into account the distance between nodes and assume that all the packets arriving at any of the intermediate network nodes will arrive at that node at the same time.
3. All intermediate network nodes are connected with both input and output edges.

4.3 Communication channels

The edges between various nodes, as described above, represent communication channels. Ideally, communication channels are error-free, however, this is not always the case, and in most practical scenarios, messages are altered by noise [9]. We will use two channel-models to model this behaviour, a Binary Symmetric Channel (BSC) and a Gilbert-Elliot (model) (GE).

4.3.1 Binary Symmetric Channel

According to [9], the BSC is often used to model communication channels in information theory since it is one of the simplest noisy channels. It is a simplistic model that can transmit one of two symbols, 1 or 0. The symbols transmitted over the network have a crossover probability p and extrapolating from that, a probability $p - 1$ to remain unchanged as seen in Figure 4.2.

It can be assumed that $0 \leq p \leq 0.5$ so more bits are not changed than stays unchanged. Even though only bit symbols of the set $\{0, 1\}$ can be sent over a BSC, we can still implement this channel model for larger Galois field sizes since all Galois field symbols can be represented as binary numbers.

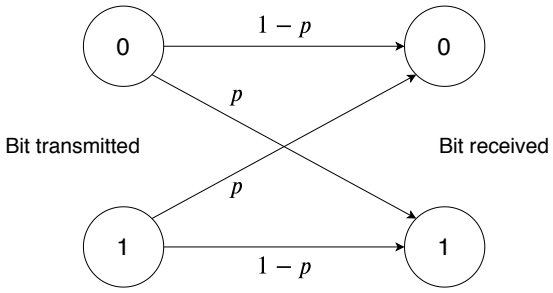

Figure 4.2: Graphical representation of a BSC and the probability of error

4.3.2 Gilbert-Elliot channel model

The GE is a channel model that simulates burst errors. The channel, proposed by Gilbert [22] and Elliot [23], is based on a Markov chain of two states, one for normal channel behaviour (N) and one for a burst channel (B).

In the N-state, the channel operates as a standard BSC, as discussed in Section 4.3.1 above, with a probability P_N of a bit error occurring. The state of the channel stays in the N-state with probability $1 - \sigma$ and can change to burst errors with probability σ. In the B-state, errors occur with a probability of P_B. Since burst errors cause bit errors to occur more frequently, $P_N < P_B$. The channel stays in the B-state with probability $1 - \kappa$ and can again change from the B-state to the N-state with probability κ.

A graphical representation of this channel model can be seen in Figure 4.3. During the B-state, the probability of a bit flip is higher than with the N-state, creating a more noisy channel. The duration and frequency of these burst errors can be adjusted by adjusting κ and σ, respectively. As κ increases, bursts are on average shorter, and as σ increases, bursts have a higher probability of occurring and are, therefore, more frequent.

4.4 Error correction for RLNC

According to [6], when using RLNC as coding technique, one of the most prominent problems with the technique is error propagation. Since data packets are combined

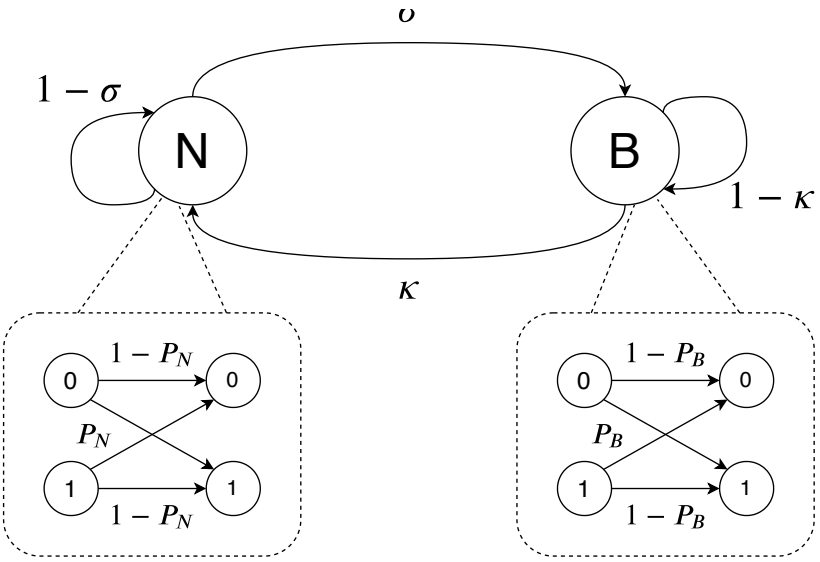

Figure 4.3: Graphical representation of a GE channel's Markov chain and channel states

together through linear operations, errors that occur early on can be combined multiple times, potentially corrupting several data packets by the time the packets reach the receiver. Concatenating a FEC code, specifically Reed-Solomon (RS) codes, with RLNC is often used to lessen the propagation of errors [9], [6], [20] and [24].

Encoding using an FEC, data can be encoded at the source node using an outer-code and an interleaver before even being encoded by RLNC encoding. The data is then sent through the network, and at the destination node, data is first decoded using RLNC. Then the FEC is decoded, by first decoding the interleaver and then the outer-code. The encoding and decoding orders can be seen in Figure 4.4.

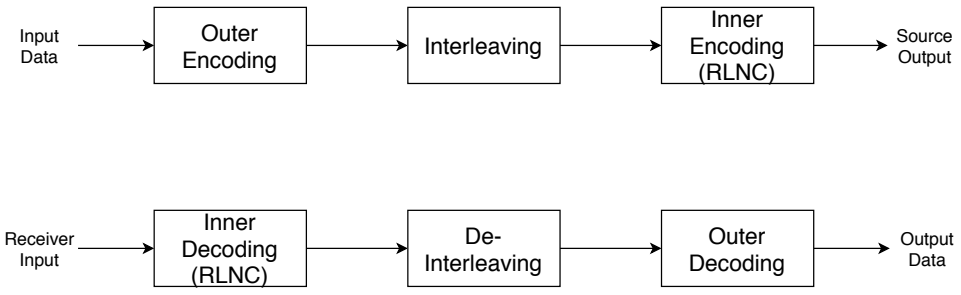

Figure 4.4: Concatenated encoding (above) and decoding (below) processes

For an outer code, we will use Reed-Solomon codes as in [18]. Reed-Solomon codes are block-based, linear (n, k)-codes. In other words, k data bits can be used to generate

$k - n$ parity bits, resulting in a final code of n bits long.

The minimum distance, d, (Singleton bound) of an (n,k) linear block code is upper bound by $d \leq n - k + 1$ with an error correction capability of $d/2$. According to [25] RS-codes can achieve the maximum of the bound and, therefore, have an error correction capability as seen in Equation (4.1):

$$d = \frac{n - k + 1}{2} \tag{4.1}$$

If for example, a packet contains k data bits, $n - k$ parity bits can be added at the end of the data bits to create a packet of n bits as seen in Figure 4.5.

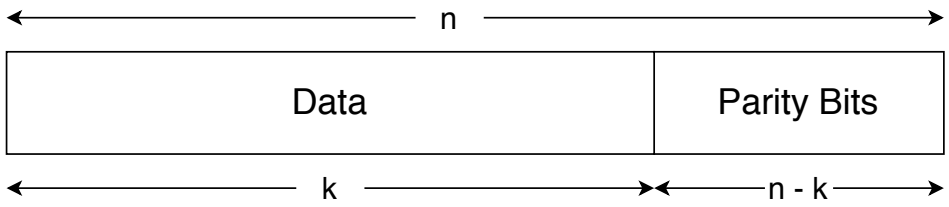

Figure 4.5: Packet structure for (n,k) Reed-Solomon encoded packet

4.5 Network error correction from Matrix Network Coding

In his seminal paper on MNC, [7], Kim *et al.* first proposed that any additional packets, $h > N$ received at the destination node can be used for error correction purposes. In this case the G-matrix is an $N \cdot p \times h \cdot p$-matrix, as discussed in Section 3.3. These additional columns have to be linearly independent of the others, thus increasing the rank of the matrix [26].

[7] discussed several decoding techniques for MNC. Different methods can be used for networks with known and unknown topologies. If the network topology is known and

unchanging, the coefficients at intermediate nodes can be chosen with that foreknowledge to create a G-matrix that is decoded with low complexity and high probability of decoding.

For this study, however, as with [7] and [20], we will use networks with an unknown topology, as is the norm for mobile networks. The networks used in this study was already discussed in Section 4.2. There are various methods to decode linear block codes, and much research has been done, such as in [27] and [28]. These decoding methods assume identical constituent vertical codes that cannot be assumed for MNC [7]. [7] determined that the task of pre-constructing the G-matrix without knowing the network topology is quite complex, and it has to be assumed that the G-matrix is entirely random.

Trellis representation

Since MNC is a finite field linear code, therefore it has a trellis representation [29], [30]. MNC could, therefore, be decoded using its trellis representation. Kim *et al.* evaluated this and determined that there are, at most $min(2^{Np}, 2^{(M-N)p})$ states [29], this would most often result in using $(M-N)p$. Larger values for p and large differences between M and N give better error correction abilities but will cause the method to be too complex [7].

4.5.1 Sphere decoding for Gaussian noise

For networks where channel errors are modelled with Gaussian noise, i.e., AWGN, Rayleigh or Rician fading channels, Kim *et al.* advocated and used a sphere decoder [28] using list decoding.

To implement this method, a parity check matrix, H is created. This is done using the generator matrix G. To create H, G is manipulated, through swapping columns and Gaussian elimination the matrix, to have the form:

$$G = [I|D] \tag{4.2}$$

In Equation (4.2), I is a $N \times N$ identity matrix and D a $N \times (M - N)$ matrix. The order in which columns are swapped to achieve the identity matrix has to be noted and used to swap the columns in the parity check matrix as well [7], [31].

Using D, H can be determined as seen in Equation (4.3):

$$H = \left[-D^T|I_{M-N}\right] \tag{4.3}$$

Using H, an algorithm, outlined in [7] is used to create the log-likelihoods of all the elements of the output packets.

This process is not, even though it has a high probability of decoding, energy-efficient; since it follows an iterative process. [7] proposed a less complicated and simplified method, Coset decoding, for BSCs that will be more energy efficient.

4.5.2 Coset decoding for MNC

Coset decoding can be low on energy if codes with simple encoders are used at the source. More energy can be saved if hard decision decoding is used instead of soft-decision decoding.

We used [7] for the steps to implement Coset decoding since the steps are listed in the article. The method uses any additional packets received to do error correction on the received data, $\boldsymbol{R}(j)$ for $j = 1, 2, ..., q$.

For this method we also use the parity check matrix, H as determined using Equations (4.2) and (4.3). Using H, a codebook C of Coset leaders can be created. The codebook contains a list of the $2^{p(K-N)}$ cosets $\boldsymbol{x}_i + C$. $\boldsymbol{x}_i$ is a binary string with minimal Hamming weight in the coset for $i = 1, 2, ..., 2^{p(K-N)}$.

Looping through the data packets for $j = 1, 2, ..., q$:

- Calculate the syndrome vector $s = \boldsymbol{R}(j)H^T$.
- Use s to find a unique vector $\boldsymbol{x}_*$ from $\boldsymbol{x}_i$ for $i = 1, 2, ..., 2^{p(K-N)}$ so that $\boldsymbol{R}(j)H^T = \boldsymbol{x}_*H^T$.
- $\mathcal{R} = \boldsymbol{R}(j) + \boldsymbol{x}_*$ is calculated as the codeword of G.
- from $\mathcal{R}$ we can calculate the input vector, the estimated $\tilde{\boldsymbol{P}}(j)$ that when encoded with G produces $\boldsymbol{R}(j)$.

4.6 Conclusion

ER-graphs can be used to create randomly connected networks of any given size. Because of their random connections, they mimic real-world networks and work well for simulations. For network edges, we discussed two methods of creating edges with network errors. The first is BSC's that create random, evenly distributed errors. The second network edge is the GE channel model that is a simple BSC with random burst errors.

To correct these errors while implementing RLNC, we can concatenate RLNC with a simple FEC scheme, such as Reed-Solomon codes. For MNC, we briefly discussed decoding methods that use the additional data, that is not needed for decoding, to correct errors. The most prominent method is Coset decoding that can be used for BSC channels.

Chapter 5

Implementation

In this chapter, the design and implementation of four experiments will be discussed. We also discuss how the experimental setup is verified, and the test implementation validated

5.1 Introduction

Four tests were designed in order to answer the research question and reach the research objectives posed in Sections 1.3 and 1.4 respectively. The first experiment was designed to improve on the decodability of MNC by increasing invertibility of the G-matrix. The second test looks at a particular case in which the output encoding matrices of intermediate nodes are guaranteed invertible. The third test will determine what the effect of network depth is on the propagation of errors through an MNC network, and finally, the effect of burst errors will be tested on the network coding technique.

In conclusion to this chapter, all the tests and implementations thereof have to be verified and validated.

5.2 Invertibility improvement

5.2.1 Motivation

For NC, each encoding factor can be chosen at random at each of the intermediate network nodes. By selectively choosing the encoding factors, we can alter the probability of decoding. This was proposed in [7] and [18] as a method to improve MNC. [7] specifically chose the encoding factors at each intermediate network node based on two factors:

- The combining coefficients are invertible;
- and all the non-zero matrix coding coefficients of the output packet are invertible

The first factor is said to be only for analysis purposes, and it is never clearly stated how the second factor is achieved. There is no literature comparing the effects of selectively choosing the encoding factors in this manner compared to choosing encoding factors at random. Also, if done by iteratively choosing encoding factors until the output is invertible, it can take a very long time and is not usable for practical implementations.

We will, therefore, test different methods that might still improve the invertibility of the G-matrix but not require endless iterations to achieve.

5.2.2 Implementation

We tested two scenarios that implemented one, or both of the factors listed above in different ways. For both scenarios, ER-graphs, as discussed in Section 4.2, were created with intermediate network nodes being connected with probability δ as seen in Equation (5.1):

$$\delta = \sqrt{\frac{n}{n-1}} \tag{5.1}$$

where n is the number of intermediate network nodes.

No errors were introduced into the network since the test only evaluates the probability of data being undecodable due to an uninvertible G-matrix.

We will use two methods, as will be discussed below, to test whether we can improve invertibility of the G-matrix. The fixed parameters that were chosen to realise both the methods are listed in Table 5.1.

Table 5.1: Simulation parameters for invertibility testing

Parameter	**Value**
Intermediate network nodes	10:5:30
Number of graphs of each node count	1000
Iterations per graph	100
Field size (F_{2^m})	2
MNC p-value	2

The unmodified, RLNC- and MNC-schemes were used as the benchmark cases to compare to the two methods discussed below.

Method 1

For the first method the **source** encoding factors are compiled into a matrix as they would be in Equation (2.9) for RLNC or Equation (3.11) for MNC to create a G-matrix. This matrix was tested and recreated at random until it was found to be invertible. It is also possible to use a list of already invertible matrices to speed up the process and use less processing power. This method increases the computational load at the source node only.

Method 2

For method 2, it was required that the encoding factors, $[\beta_1, \beta_2, ..., \beta_N]$, that are the outputs of any intermediate network node, should be invertible.

This was achieved by first choosing all encoding factors, whether at the start or intermediate nodes, to be only invertible matrices. For each of the encoding factors at intermediate nodes to be chosen as invertible matrices, these factors should be of a size $p \geq 2$. This stipulation limits this method to apply only to MNC.

Computations were performed at the intermediate node to determine the output encoding factors. If the output matrices are not invertible, up to 10 iterations were done for which most cases resulted in an invertible matrix.

This method increases the computational load at all of the network nodes.

Both methods were implemented, and the results of the test can be seen in the Results chapter, Section 6.2.

5.3 Upper triangular matrices as encoding factors

5.3.1 Motivation

Kim [7] suggested using encoding matrices such that all the non-zero matrix coding coefficients of the output packet at intermediate nodes are invertible. This was attempted for a limited number of iterations using Method 2 in the previous section. Output encoding matrices are created by adding and multiplying different coefficients together as seen in Equation (3.10) in Section 3.2.4. Invertibility of a random matrix is, however, only held over multiplication but not addition [26].

We, therefore, want to investigate a small subset of matrices that are invertible and for which invertibility is sustained over both addition and multiplication. Encoding

factors can then be chosen from this subset of matrices to maintain invertibility of the output encoding matrices throughout the network.

According to [26], an upper triangular matrix is invertible if its determinant, in this case, the product of all the values on the diagonal, is not zero. Therefore for $GF(2)$, an upper triangular matrix with only 1's on the diagonal is always invertible. This matrix is known as a unit upper triangular matrix, according to [26], [32]. Equation (5.2) shows such a matrix, L, with $p = 4$, where the X's denote either 1 or 0.

$$L = \begin{bmatrix} 1 & X & X & X \\ 0 & 1 & X & X \\ 0 & 0 & 1 & X \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

From [33], it can be found that the upper triangularity of a matrix is sustained through both addition and multiplication. Therefore, if we choose all encoding factors as random upper triangular matrices with only 1's on the diagonal, we will sustain upper triangularity.

This will not sustain invertibility since it does not guarantee that there will be only 1's on diagonal of the new matrix. When looking back at Equation (3.10) in this context, we find that when adding matrices, we use a X-OR operation. Due to this operation, 1's on the diagonal can only be sustained when an uneven number of matrices are added together.

To sustain unit upper triangular matrices through a network, we must limit the networks to those that have only an uneven number of outputs to the source node and an uneven number of inputs to any other node.

5.3.2 Implementation

To test a network with only an uneven number of outputs from the source node N and an uneven number of inputs to any other node, h and M, we will choose the test network as seen in Figure 5.1.

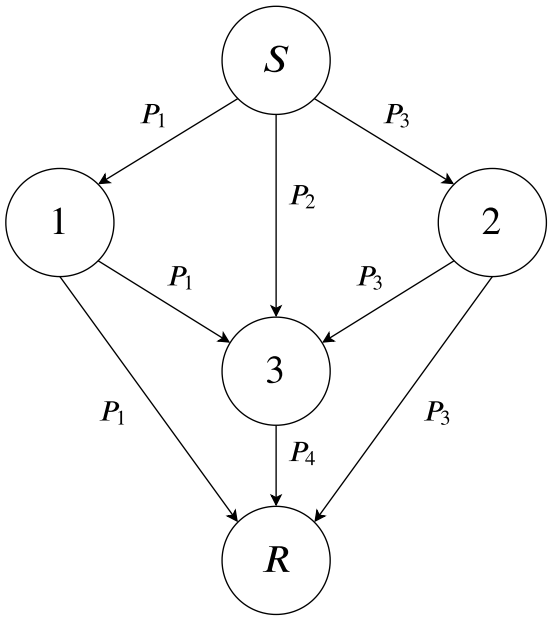

Figure 5.1: Network graph used to test the use of upper triangular matrices

In Figure 5.1, we see a network that is simplistic yet meets the requirements to have an uneven number of inputs and source outputs. The network also grants the opportunity for network coding at node 3, where three packets are combined into one.

Using this network, we will test the theory of invertible upper triangular matrices, along with methods 1 and 2 from the previous section against choosing the encoding factors at random. For each method, 10 000 graphs will be used with encoding matrix size 2×2, given $p = 2$ and field size $GF(2)$.

The results of the test can be seen in Section 6.3 in the next chapter.

From the first experiment on invertibility improvement, we saw that in order to improve the invertibility of the G-matrix, we eliminated some encoding factors. In doing so, we also lower the number of possible combinations created through network cod-

ing, therefore limiting the number of eventual G-matrices. Since MNC has intrinsic error correction capabilities, [7], we will now evaluate the network error correction capabilities of a system with only invertible matrices as encoding matrices over different factors in Sections 5.4 and 5.5.

5.4 The influence of network depth on error propagation

After first considering error-free graphs in Sections 5.2 and 5.3, we move on to a more real-world scenario where network edges are error-prone. To test the effect of errors on different network depths, we must first add errors to the still error-free graphs used in the previous section. For the error model, we used a BSC model, as discussed in Section 4.3.1.

In [7], a Binary Phase-Shift Keying (BPSK) modulation scheme with Gaussian noise is used with a Signal to Noise Ratio (SNR) of $SNR = 1 \rightarrow 9$. For a BPSK channel, the relation between $SNR(dB)$ and the normalised signal to noise ratio, $\frac{E_b}{N_0}$, is $10\log(SNR(dB)) = \frac{E_b}{N_0}$ [34]. The BSC function in Matlab uses a probability of error, P_b, denoted by a number between 0 and 1 [35] not SNR or $\frac{E_b}{N_0}$. In order to get comparative results, we can convert SNR to Bit Error Rate (BER) using Equation (5.3) where $erfc$ is the cumulative error function [36]:

$$\begin{aligned} P_b &= \frac{1}{2}\text{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) \\ &\frac{1}{2}\text{erfc}\left(\sqrt{10^{SNR/10}}\right) \end{aligned} \tag{5.3}$$

The BER resulting from a $SNR(dB) = 1 \rightarrow 9$ can be seen in Figure 5.2.

These BER values were used to generate errors in a BSC. The following experiment was done to test the effect of network depth on error propagation through an MNC system.

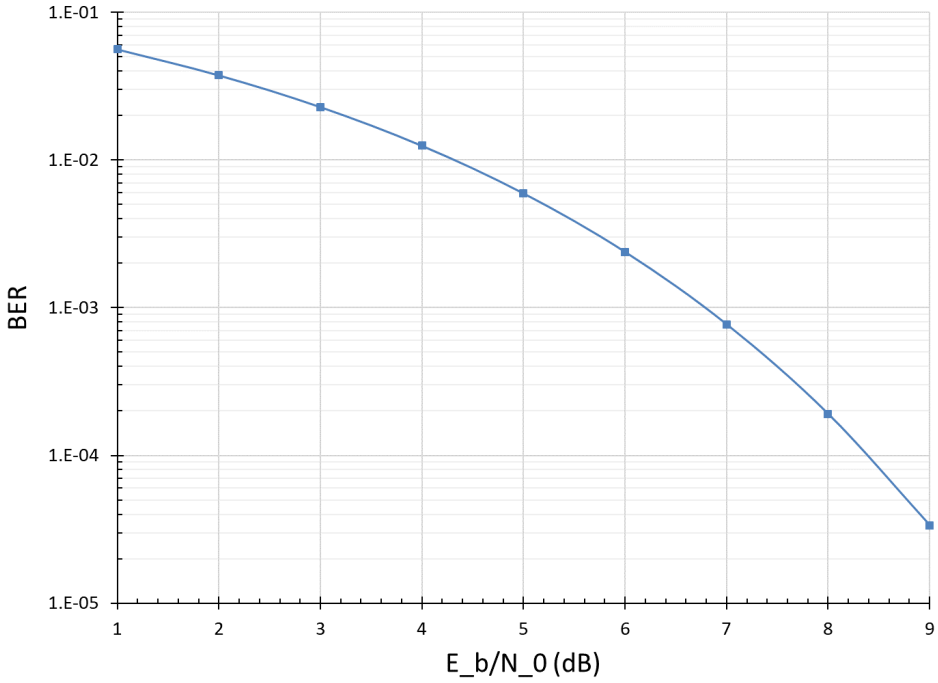

Figure 5.2: BER against SNR(dB) for BPSK

5.4.1 Implementation

For different network depths, from 10 to 25 with intervals of 5, 1000 random ER-graphs where created. All of the graphs have the same number of input edges, N, output edges, M, and a network connectivity probability of 0.65. Graphs that did not reach a $min - cut$ of N where discarded. For each input node, $L = 256$, data bits where used. For all the runs, MNC was used with a fixed field size of $GF(2)$ and the size of encoding matrices were kept as 2×2. The only variable is the network depth, varied by changing the number of intermediate nodes. Each of the intermediate network edges is a BSC with an SNR of 8 and $9dB$.

All of the parameters discussed can be seen summarised in Table 5.2.

The results of the test are discussed in Section 6.4.

Table 5.2: Simulation parameters used to determine the effect of network depth on error propogation

Parameter	Value
Encoding type	MNC
Field size (F_{2^m})	1
MNC p-value	2
Number of graphs	1000
Network connectivity	0.65
N	3
M	4
Intermediate network nodes	10 : 5 : 25
Uncoded packet length (bits)	256
Channel type	BSC
SNR for errors	8 and $9dB$

5.5 Influence of burst errors

In order to test the effect of the intensity of burst errors on MNC, we will compare the number of erroneous packets received at the destination node of different error-prone networks. As discussed in Chapter 4, there are different error correction methods available for RLNC and MNC that are implemented on error-prone channels. For MNC, we use Coset decoding, as discussed in Section 4.5 and implemented as seen below.

5.5.1 Coset decoding

To implement Coset decoding, Matlab functions for Coset leaders, discussed in Section 4.5.2, were implemented that can be used in conjunction with the H-matrix to determine errors in the data packets.

5.5.2 Burst errors

We modelled burst errors using a GE-channel model. The channel model is fully discussed in Section 4.3.2. To implement this channel model, we used two different states, normal (N) and burst (B). If, for example, we choose the probability of burst errors occurring to be $\sigma = 10\%$. It would mean, the channel has the highest probability of $\kappa = 0.9$ to stay in N while there is a 10% chance to move to B. When in B, the channel would tend to go back to normal with a 90% chance. This example can be visually seen in Figure 5.3.

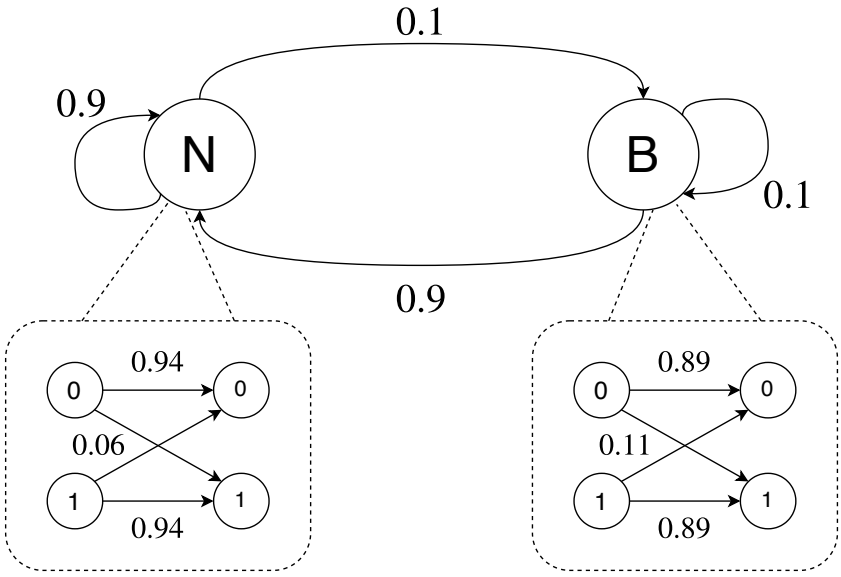

Figure 5.3: Implementation parameters of a GE channel's Markov chain and channel states

In Figure 5.3, it can also be seen that the two states have two different error probabilities. Using Equation (5.3), we can calculate that the normal probability of errors in the N state at SNR(dB) $= 1$ is 0.06. For example, we will create burst errors of double the error probability of the N-state. In this case, the probability errors in the B-state is 0.11, as seen in Figure 5.3.

5.5.3 Motivation for the test

For most previous tests performed with MNC, only uniform error channels, such as the BSC, or Additive White Gaussian Noise (AWGN) channels where used for experimental implementation [7], [18].

The effect of burst errors has not been tested, while applying MNC, before. This test will, therefore, be used to evaluate the performance of the intrinsic error-correcting abilities of MNC over channels that exhibit burst error behaviour.

5.5.4 Implementation

For this experiment, we created 1000 graphs of 10 nodes each. The graphs are created randomly as ER-graphs while the channels used are GE-channel models. The GE-channel simulates burst errors on an unaltered BSC, as discussed in Section 4.2.

Several factors where kept constant during these experiments. The number of input edges, $N = 3$, and output edges are $M = 4$. The field size for MNC is kept at $GF(2)$ and the encoding factors were chosen as 4×4 matrices, meaning $p = 4$.

For the errors, the unaltered BSC had a noise level of $5 \rightarrow 11dB$, which is the range from which data can be decoded using a BSC. This was varied for the GE-channel to determine the influence of burst intensity. Three different scenarios were created in addition to the unaltered BSC. The probability of bursts occurring was kept the same at $\sigma = 10\%$ for all the scenarios. The variable factor is the burst intensity which was varied between 10-times, 100-times and 1000-times the intensity of the unaltered BSC.

The parameters discussed above can be seen summarised in Table 5.3.

The results obtained from these experiments will be discussed in Section 6.5.

Table 5.3: Simulation parameters used to determine the effect of burst errors for different network depths on error propogation

Parameter	**Value**
Encoding type	MNC with Coset decoding
Field size (F_{2^m})	1
MNC p-value	4
Number of graphs	1000
N	3
M	4
Intermediate network nodes	10
Uncoded packet length (bits)	256
Channel type	BSC and GE
Burst intensity	$\times 10, \times 100, \times 1000$
SNR for errors	$1 \rightarrow 9dB$

5.6 Verification and validation

According to Sargent [37], in order to trust the validity of the results created, we must assure that programming and implementation was done correctly. We must, therefore, verify that RLNC and MNC were implemented correctly. Validation of the program consists of proving that the results can be used to compare the results to existing research and that the implementation will answer the posed questions in Section 1.3.

5.6.1 Verification

In order to verify the MATLAB simulation models, hand-written examples following the discussion of RLNC and MNC in Chapters 2 and 3 where implemented. Two such examples were also discussed in-depth in Sections 2.4 and 3.5. For these tests, the butterfly network, input data and encoding factors where identically replicated from the hand-written examples into MATLAB. The data sent into the network and the data decoded at the destination nodes where identical.

As mentioned in Chapter 3, MNC is equal to RLNC when $p = 1$. This test was used to further verify the validity of MNC by running the example for RLNC, discussed in Section 2.4, on the MNC simulation. The same results were achieved for both the network coding methods.

To verify that the implementation was done correctly for random graphs with random encoding coefficients, random ER-graphs were implemented, and all the data and encoding factors were randomised. This test was done before any bit errors were implemented. No packets were decoded incorrectly while running several tests over different encoding factor sizes and Galois field sizes; verifying again that the implementation was done correctly.

To verify the implementation of the Reed-Solomon outer code for RLNC and the Coset decoding for MNC, the methods were first run without any errors occurring. With the Reed-Solomon code and Coset decoding implemented for RLNC and MNC respectively, no bit errors were introduced, and no packets were decoded incorrectly.

Then bit errors where introduced and a higher decoding rate was obtained with the implementation of the FEC codes, compared to when the FEC codes were not implemented.

5.6.2 Validation

For the three different experiments, we will evaluate whether the experiments are designed in such a way that they can answer the research question and reach the objectives set in Sections 1.3 and 1.4 respectively. We will then compare results from the experimental setup with that of existing work.

Evaluation of experimental setups:

In each experiment, specific care was taken to vary as few parameters as possible. This was done to ensure that the test results are only dependent on the parameter being tested. For example, in the experiment on network depth, only the number of intermediate network nodes were varied. In all of the experiments, several random graphs were created. This was done to eliminate the effect that one specific network connectivity pattern can have on the results.

Comparison to existing work:

In the original simulation done by Kim *et al.* in [7], MNC was modeled with a sphere decoding algorithm as discussed in Section 4.5.1. One of the tests in [7] was performed to test the extent to which the number of input edges at the output node influence the results. For this experiment 3, 4 and 5 inputs where evaluated. The experimental graph discussed in [7] can be seen in Figure 5.4 below:

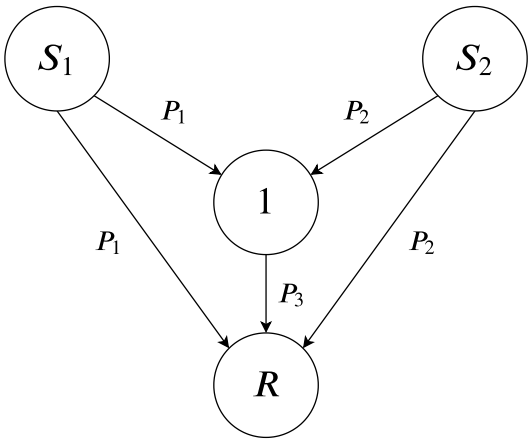

Figure 5.4: Network graph used in [7] for simulations

This graph has four fixed nodes, of which two are source nodes, and one is a destination node. Only one intermediate node generates the opportunity for network coding. The sources both transmit **uncoded** packets, P_1 and P_2 respectively. MNC or RLNC is only performed at the intermediate node to create P_3. All the edges of the net-

work are subject to Gaussian noise, specifically AWGN, with a signal to noise ratio of $SNR(dB) = 1 \rightarrow 9$.

Since our ER-graphs only have one input and one output node, the network in Figure 5.4 was slightly adapted as seen in Figure 5.5.

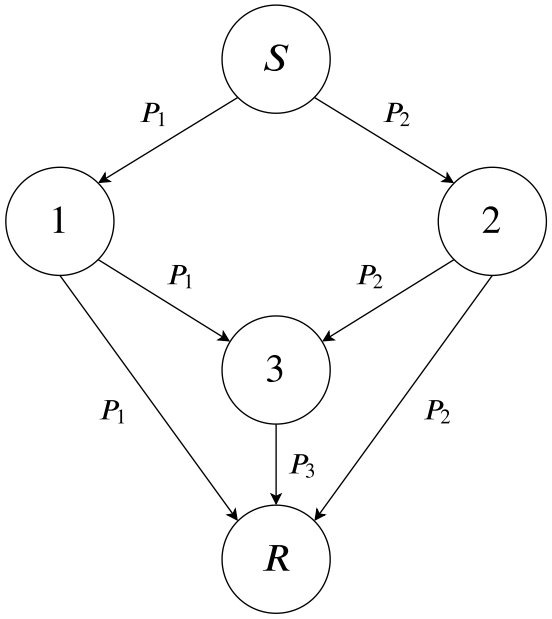

Figure 5.5: New graph to represent the graph used in Figure 5.4

As seen in Figure 5.5, a new input node is added. This, effectively, did not change anything to the setup since two different packets, P_1 and P_2 reach nodes 1 and 2 respectively. These packets are forwarded from nodes 1 and 2 as would be the case for 5.4. For our experiment, source encoding is performed at the source node. This adds an extra layer of encoding to the packets received at the destination node.

Even though the specific graphs with $M = 3 \rightarrow 5$ were not discussed, we used graphs similar to that seen in Figure 5.5 to recreate the results of [7]. For all the experiments the size of the encoding matrices where kept $p = 2$. The parameters used for the MNC implementation of [7]s experiment, as understood, can be seen summarised in Table 5.4.

The biggest change in this test from that in [7] is the channel type. Instead of Gaussian noise, a BSC was used for this experiment.

Table 5.4: Simulation parameters used by Kim *et al.* in [7]

Parameter	**Value**
Encoding and decoding type	MNC with Coset Decoding
Field size (F_{2^m})	1
MNC p-value	2
Number of graphs	100×1000
N	2
M	$3 \rightarrow 5$
Uncoded packet length (bits)	16
Errors	Gausian for BPSK
SNR for errors	$1 \rightarrow 8(dB)$

Table 5.5: Simulation parameters used to recreate previous results with a BSC

Parameter	**Value**
Encoding and decoding type	MNC with Coset Decoding
Field size (F_{2^m})	1
MNC p-value	2
Number of graphs	10 000
N	2
M	$3 \rightarrow 5$
Uncoded packet length (bits)	16
Errors	BSC
SNR for errors	$1 \rightarrow 8(dB)$

The conversion of SNR to BER is discussed in Section 5.4. Since the channel type is changed, the decoding method for MNC has to be changed from the sphere decoding algorithm, discussed in Section 4.5.1, to Coset decoding, discussed in Section 4.5.

The simulation parameters used for the BSC test can be seen in Table 5.5, while the results obtained from the test can be seen in Figure 5.6.

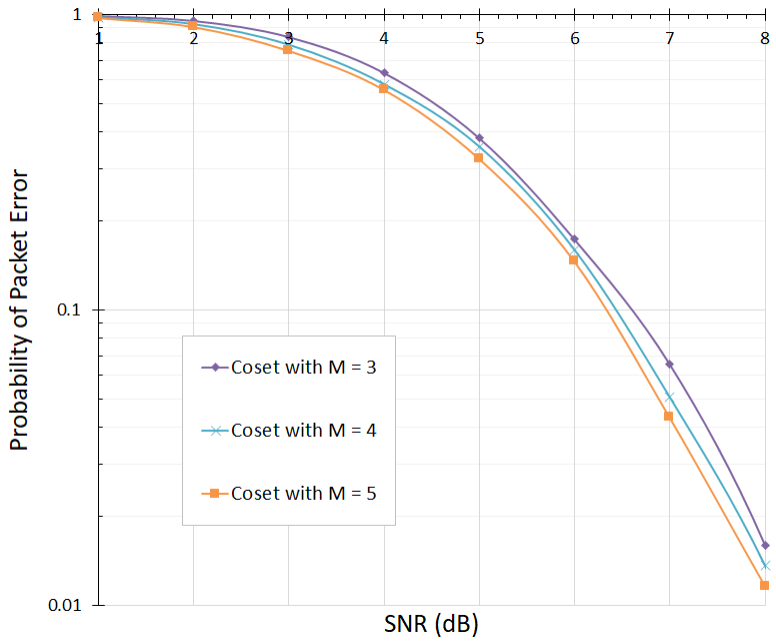

Figure 5.6: Probability of packets at destination node containing errors for MNC over BSCs

Figure 5.6 shows the probability that the packets, obtained at the destination node after Coset decoding, contain errors over different SNR values. In the graph, it can be seen that as the SNR increases, the probability of packet errors goes down. Also seen in Figure 5.6 is that more input edges at the output node result in more decodable packets and fewer errors; since more packets can be used for error correction.

The results, shown in Figure 5.6, are similar to those in [7]. The curves follow the same order with $M = 3$ showing the most errors and $M = 5$ showing the least amount of errors. The curves also follow a downward slope, showing fewer errors as the SNR decreases. There are some discrepancies in values between the two experiments. These discrepancies can be attributed to the differences between the graphs used in the two experiments, the use of Coset decoding, that uses hard decision decoding, rather than the soft decision sphere decoding of [7], and the use of source encoding.

5.7 Conclusion

In this chapter, we discussed the implementation of four experimental designs. We discussed both the implementation of certain parts of the experiments in MATLAB and how specific parameters where chosen in order to answer the research question and reach the objectives stated in Chapter 1. We also discussed how the implementation and setup were verified and validated.

From the verification and validation exercises shown in this chapter, we conclude that the experimental platform was implemented correctly and is suitable to investigate the research questions. The results of the experiments can be seen compiled in the following chapter.

Chapter 6

Simulation Results

The results of all the experiments are discussed in this chapter. Include the results of doing testing on different methods to improve the invertibility of the G-matrix. We also show the effect the network depth has on error propagation. Finally, the results of implementing a GE channel model with different burst intensities are shown.

6.1 Introduction

We will now present the results from the implementation discussed in the previous chapter. This shows the result of four experiments, the first to determine the impact of choosing different encoding factors of the invertibility of the G-matrix. In contrast, the second experiment shows the impact of choosing only invertible upper triangular matrices as the encoding matrices for MNC. The third experiment shows the impact of error propagation on networks of different sizes and the last experiment shows the impact of different burst error sizes on networks implementing MNC.

6.2 Invertibility improvement

For this test, we discretely chose encoding factors through two different methods as described in Section 5.2. The test is valuable since it investigates small fields and limitedly connected networks found in real-world scenarios for RLNC and MNC.

Figure 6.1 shows a Complimentary Cumulative Distribution Function (CCDF) of the percentage invertible *G*-matrices received at the destination node of randomly created graphs. From Figure 6.1 we see that 80% of random networks that implement the unmodified RLNC-scheme have only a 40% probability of correctly decoding received packets at the destination node. Similarly, there is an increased probability from 51% to 90% that a random network that implements RLNC with Method 1 will have a 50% probability of correct decoding.

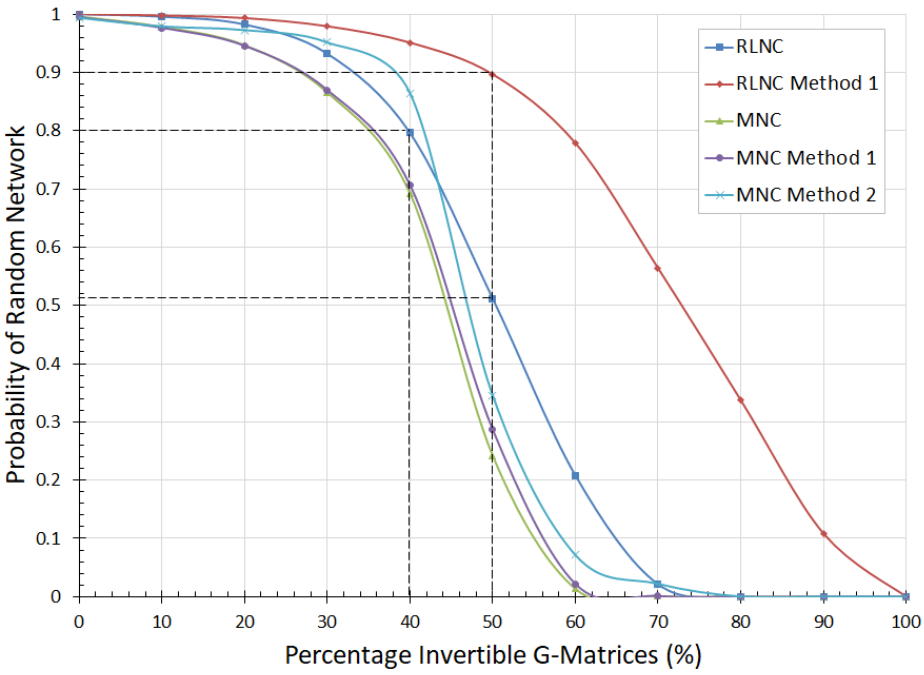

Figure 6.1: Probability of a randomly generated network graph to have a certain percentage of invertible G-matrices, expressed as a CCDF

The average, median and standard deviation of the number of invertible *G*-matrices for each method in Figure 6.1 can be seen summarised in Table 6.1:

Table 6.1: Statistical parameters for the decodability of randomly generated network graphs

	Average	**Median**	**Standard Deviation**
RLNC	50.0	51	12.2
RLNC Method 1	71.5	73	16.4
MNC	42.8	45	11.5
MNC Method 1	43.5	46	11.8
MNC Method 2	47.5	48	10.8

Consider the graphs for MNC in Figure 6.1. Both Methods 1 and 2 show some improvement in decodability over the unimproved case, although not as much as with RLNC. However, not all graphs showed an improvement, with only 65% of randomly selected networks showing some improvement. Closer investigation of these random networks that show improvement is shown in Figure 6.2.

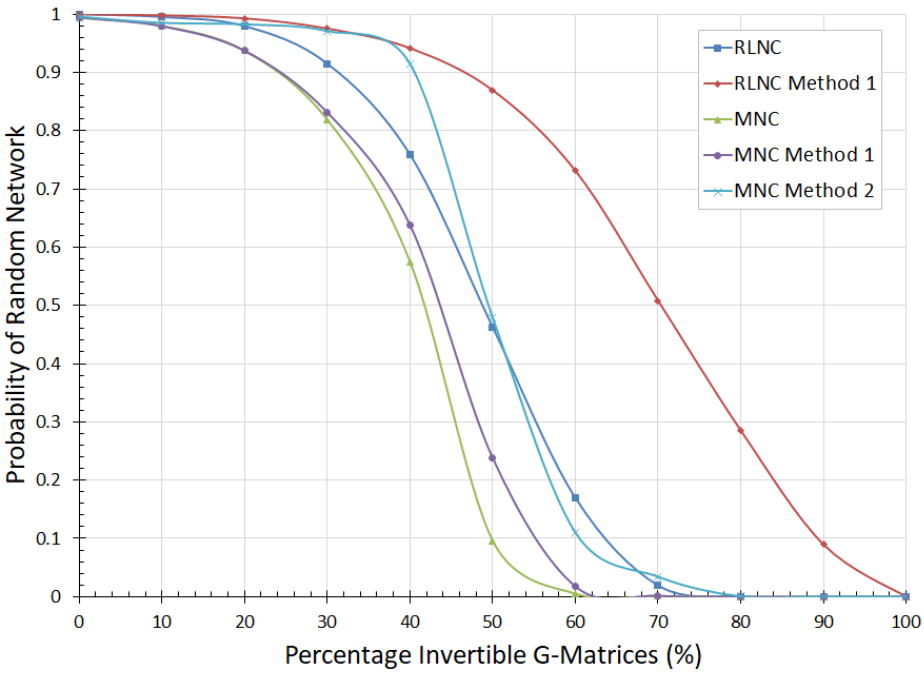

Figure 6.2: Probability of a subset of network graphs, where method 2 shows improvement, to have a certain percentage of invertible G-matrices, expressed as a CCDF

From Figure 6.2, it is seen that both Methods 1 and 2 improve the decodability of MNC.

In the case of Method 2, the improvement shows that on average, in 55% of the cases that MNC has a better probability of successful decoding than the baseline RLNC.

Figure 6.3 shows only the 15% of random networks that have more than 50% improvement in decoding probability.

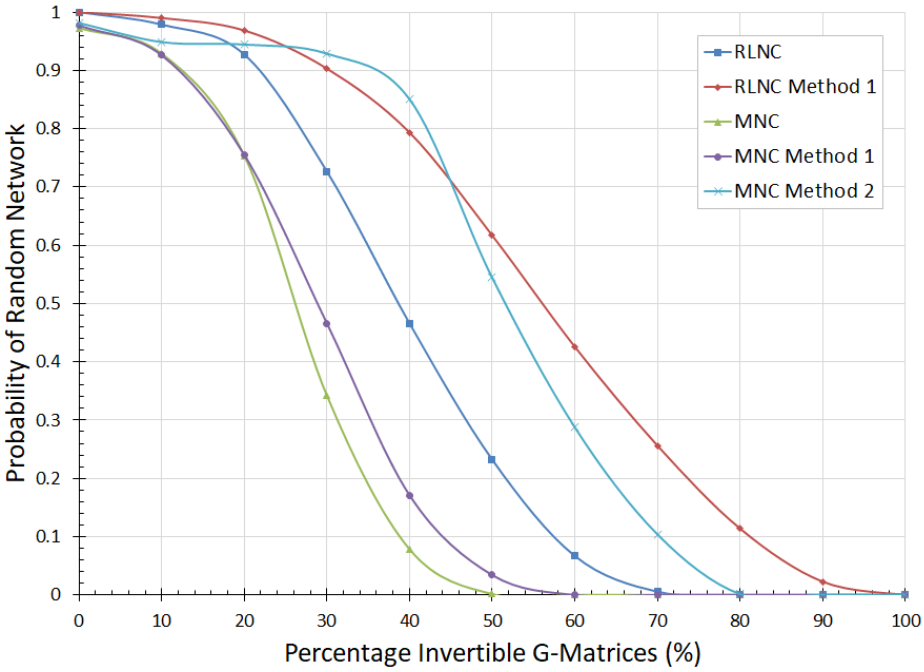

Figure 6.3: Probability of a subset of network graphs to have a certain percentage of invertible G-matrices, where method 2 shows an improvement of 50% or more, expressed as a CCDF

From Figure 6.3, it is seen that in some of the cases that have more than 50% improvement from their baseline implementations, MNC with Method 2 even improves on RLNC with Method 2.

The statistics from Figure 6.3 can be seen in Table 6.2.

It can be noted that the average percentage invertible *G*-matrices for MNC method 2 improves and all the other values are lower than in Table 6.1. From this, we can see that Method 2 works exceptionally well for graphs that would otherwise have a lower than average number of invertible G-matrices.

Table 6.2: Statistical information derived from Figure 6.3 to describe the different methods analysed

	Average	**Median**	**Standard Deviation**
RLNC	39.6	40	13.9
RLNC Method 1	56.4	57	19.2
MNC	26.2	27	10.4
MNC Method 1	28.8	30	12.3
MNC Method 2	51.6	51	16.2

Figure 6.4 shows how the number of invertible G-matrices, and, therefore, the decodability of the network changes for different numbers of intermediate network nodes.

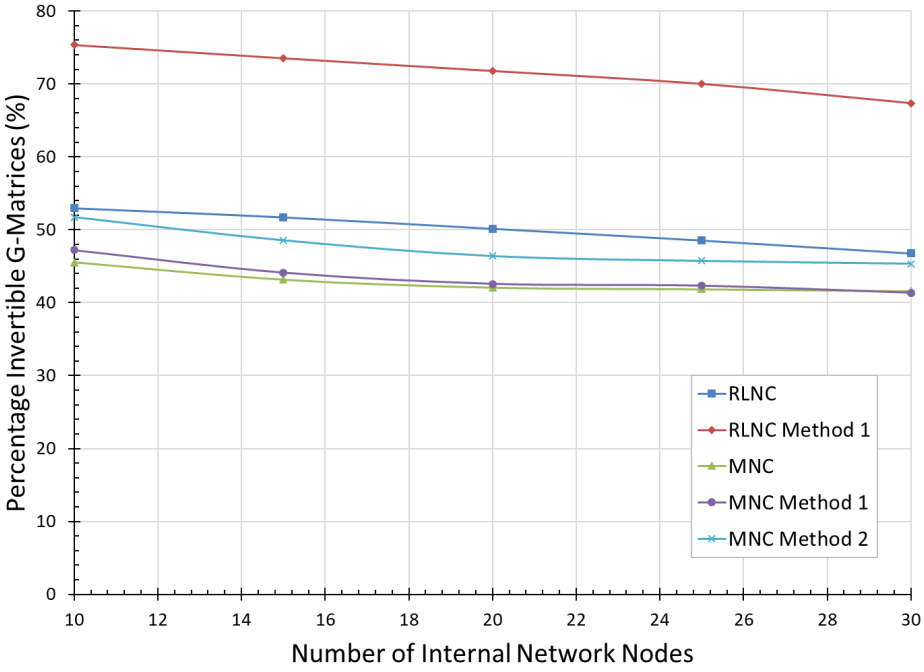

Figure 6.4: Percentage invertible G-matrices for graphs with various numbers of intermediate network nodes

It can be seen that the percentage invertible *G*-matrices follow the same order, with RLNC Method 1 having the most and MNC having the lowest percentage invertible *G*-matrices. It can also be seen that the percentage invertible *G*-matrices also decrease as the number of intermediate network nodes increases.

6.3 Upper triangular matrices as encoding factors

For this experiment, we compared three different methods of selectively choosing encoding matrices to an unaltered MNC scheme. The methods were all tested on the network seen in Figure 5.1. Method 1 and 2 from the previous section were implemented on MNC along with the new method where only invertible upper triangular matrices were used as encoding matrices. These three methods were then compared to unaltered MNC, i.e., MNC for which encoding matrices were chosen at random.

The increased number of invertible G-matrices for each of the three methods when compared to unaltered MNC, can be seen in Figure 6.5 below.

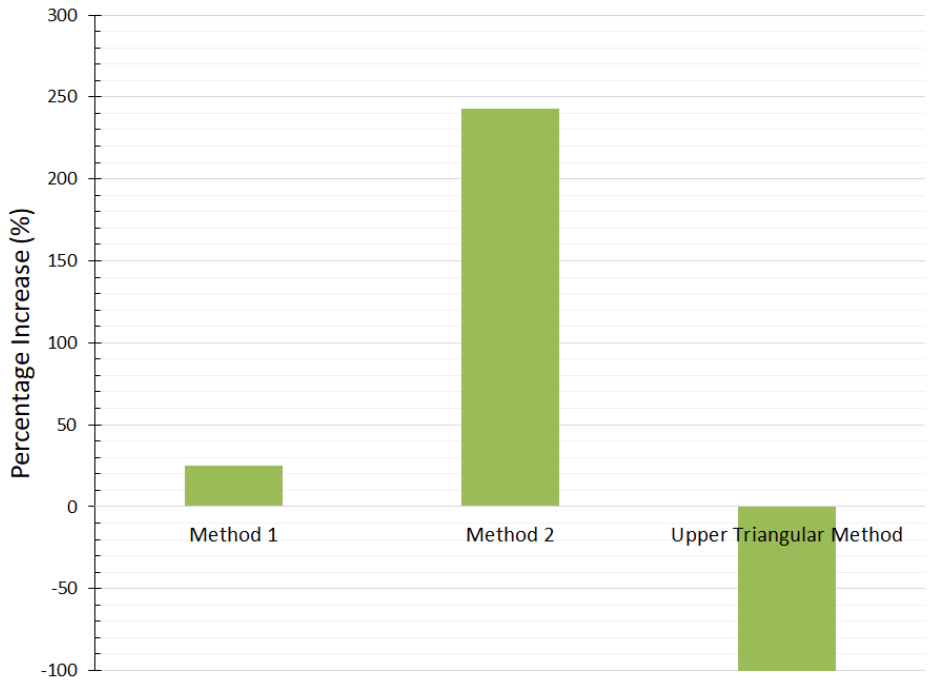

Figure 6.5: Percentage increase in the number of invertible G-matrices, by selectively choosing encoding factors, from using random encoding factors

It is seen in Figure 6.5 that both Method 1 and Method 2 show improvement from randomly choosing encoding matrices using the network seen in Figure 5.1. The improvement in the number of invertible G-matrices using Method 2 is significant at almost 250%. These results correspond to the results seen in Section 6.2.

Using only upper triangular matrices resulted in **NO** invertible G-matrices out of 10 000

matrices. The upper triangular method is, however, only a specific case of Method 2 where instead of iterating to create invertible encoding matrices at the outputs of intermediate nodes, we chose only encoding matrices that would ensure this outcome.

When evaluating the G-matrix, formed using the upper triangular method, for the specifications listed in Section 5.3, we find a matrix that is always of the form seen in Equation (6.1), where X denotes either a 1 or a 0.

$$G = \left[\begin{array}{cc|cc|cc} 1 & X & 1 & X & 1 & X \\ 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 1 & X & 1 & X & 1 & X \\ 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 1 & X & 1 & X & 1 & X \\ 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right] \tag{6.1}$$

Each of the intermediate matrices to the G-matrix is upper triangular with only 1's on the diagonal, as expected. Each of the 9 matrices inside of G is always invertible; however, G is never found to be invertible.

The reason for this phenomenon is quite simple, according to Lay [26], a matrix, A is invertible if, and only if, its determinant $det(A) \neq 0$. However, the determinant of any square matrix is equal to 0, if any of the rows in the matrix are the same, i.e. not linearly independent. In Equation (6.1), we find that each of the even rows of the matrix is the exact same, resulting in the matrix always being uninvertible.

The G-matrix in Equation (6.1) consists of 9 sub-matrices. Each of the size 2×2 for $p = 2$. If p was chosen to be any larger number, the same problem would still apply, since the prerequisite of the matrices is to be upper triangular matrices with 1's on the diagonal. The final row, r_p, of each sub-matrix will always be $r_p = [r_p^1, r_p^2, ..., r_p^{p-1}, r_p^p] = [0, 0, ..., 0, 1]$. Therefore this method would not work regardless of the size of p.

The failure of the method is also independent of the size of the network, since adding another row of matrices with the same form as seen in Equation (5.2), will only result

in a G-matrix where even more rows are exactly the same.

We can see from these results that choosing all encoding matrices to be invertible upper triangular matrices, though it does result in the output encoding matrices of each intermediate node being invertible, results in a G-matrix that would never be invertible regardless of the size of the network or the size of the encoding matrices.

6.4 Network errors over network depth

After determining methods that improve on the invertibility of the G-matrix and discussing one method that would never result in invertible G-matrices, we move on to error-prone networks. The first test is to determine what the influence of network depth is on the propagation of errors through the network. The implementation of this test is discussed in Section 5.4. By applying the implementation discussed in Section 5.4, we obtained the results shown in this section.

The graph in Figure 6.6, shows the effect of the number of intermediate network nodes on the number of packets arriving at the destination node that contain errors. For this, any output data that will be undecodable, be it due to errors in the data or the G-matrix is deemed 'erroneous packets', similar to the method discussed in [7]. The result of this is that if even a single error is found in any output packet data or header, the data has a packet error.

Figure 6.6 shows the number of times packets at the destination node cannot be decoded without errors for networks of different sizes. This is all done at $SNR = 8$ and $9dB$. It should also be noted that the y-axis has a logarithmic scale.

It can be seen from Figure 6.6 that, for the same SNR(dB), the greater the number of intermediate network nodes, the higher the probability of receiving erroneous packets at the destination node. The graph does appear to flatten as the networks become more extensive and this can be seen for both $SNR = 8dB$ and $SNR = 9dB$. Therefore, the

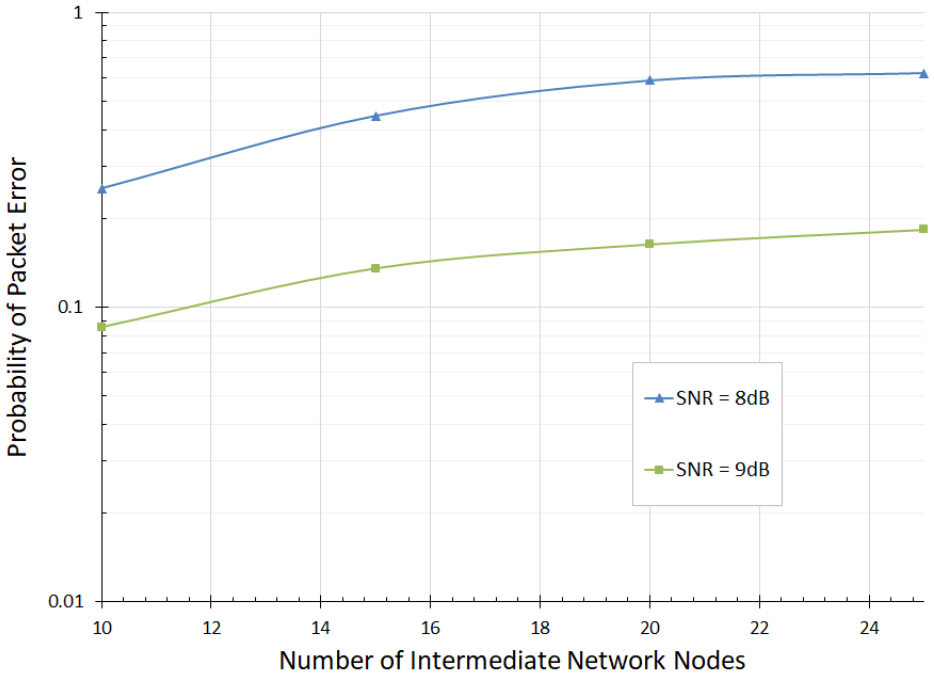

Figure 6.6: Percentage erroneously decoded packets for graphs with various numbers of intermediate network nodes

more extensive the network, the less the increase in the probability of packet errors.

6.5 Influence of burst errors

To see the effect of burst errors on MNC, different networks were created where burst errors occur 10% of the time with an error rate of $\times 10$, $\times 100$ and $\times 1000$ the original probability of the BSC. The motivation and setup for the test was discussed in Section 5.5 in the previous chapter. The results can be seen in Figure 6.7.

Figure 6.7 shows the probability of wrongly decoding packets at the output node of a network for different SNR(dB) values. The probability in this graph is linear. In the graph, four different results can be seen. The graph of $\times 1$ is equivalent to a normal BSC channel without bursts and serves as the baseline results. From there the bursts intensities get exponentially larger from $\times 10$ to $\times 100$ to $\times 1000$.

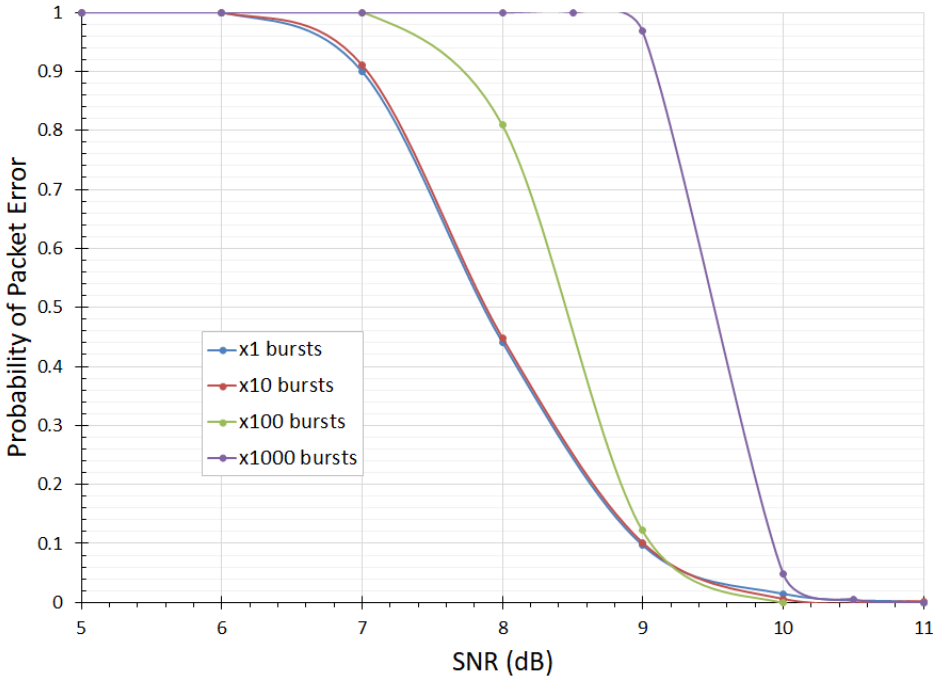

Figure 6.7: Probability of wrongly decoded packets for networks with a GE-channel model with bursts of various intensities

For all the graphs, it can be seen that for low SNR(dB), at least below 6dB, no data was decoded without errors. It can also be seen that for high SNR(dB), $SNR > 10.5$, almost all the data packets were decoded error-free regardless of the burst intensity.

The most exciting difference seen between the graphs is that the higher the burst intensity, the steeper the decline from $1 \rightarrow 0$. It can also be seen that the SNR(dB) at which some packets can be decoded without errors increases, the more intense burst errors are. The results that are shown here apply to the implementation parameters discussed in Section 5.5 and different results would be obtained with other parameters.

The intrinsic error correction of Coset decoding decreases the number of packet errors since the method corrects both data and header errors.

6.6 Conclusion

The results documented in this chapter show that selecting invertible encoding matrices at the source node results in a significant improvement in the probability of creating an invertible *G*-matrix at the source node for RLNC and a small improvement for MNC. It can also be seen that when choosing encoding matrices in such a manner that the output of intermediate nodes is invertible, there is a much more significant impact on the number of invertible G-matrices created for MNC. We also found that some graphs show a much more significant improvement from choosing only invertible matrices as output to intermediate nodes than others.

We also find that there is a slight decrease in the decodability of networks as the number of intermediate network nodes increases. In this experiment, the network connectivity is dependent on the number of network nodes to assure minimum network connectivity.

From the first experiment, we chose an even smaller subset of matrices that are invertible upper triangular matrices as encoding matrices. This resulted in the output encoding matrices of all the intermediate nodes always being invertible. The resulting *G*-matrix was, however, never invertible. We also found that the form of the resulting *G*-matrix shows that this method would always result in an uninvertible *G*-matrix, regardless of the size of the network or the size of encoding matrices.

From the results of networks with errors added to the network edges, we also find that the deeper the network is, or the more intermediate nodes the network has, the higher the error propagation through a network implementing MNC. It can, however, be seen that for networks with intermediate nodes from 20 upwards, the error propagation starts to plateau.

The final results obtained in this chapter is the influence of burst errors of different intensities on the decodability of data sent through a network implementing MNC. From the results, it can be seen that the higher burst errors are, the steeper the decrease

in wrongly decoded packets. For the network setup used in this experiment, we would recommend using MNC on networks with an SNR of 10dB or higher. The main effect of high-intensity burst errors is that the decodability of packets over SNR(dB) has a steeper decrease curve. Therefore, the same SNR(dB) of 10.5dB can be used for a BSC as for a GE-channel with x1000 burst intensity of 10% probability.

Chapter 7

Conclusion and Recommendations

This chapter concludes the dissertation, giving an overview of the entire dissertation and explicitly stating how the research question was answered, and the research objectives were met. Some suggestions and recommendations for future work are also discussed in this chapter.

7.1 Introduction

In this dissertation, we investigated whether it is possible to improve the decoding efficiency of Matrix Network Coding (MNC). MNC was often compared with another LNC technique, Random Linear Network Coding (RLNC) since RLNC is one of the most used LNC techniques. The two techniques, RLNC and MNC, were discussed throughout the literature study in Chapters 2 and 3, respectively. Chapter 4 presents an in-depth discussion on networks and transmission through networks, and how error correction techniques can be implemented for both RLNC and MNC to reduce the effect of errors.

After the literature study, the implementation of specific tests was discussed in Chapter

5. These tests were used to answer the research question and objectives discussed in Chapter 1. The methods were verified and validated, also in Chapter 5, and then implemented. The results of the tests are given in Chapter 6.

7.2 Research overview

Invertibility improvement

For the first experiment, two different methods were tested to evaluate their impact on increasing the likelihood of invertibility of the *G*-matrix for randomly created networks when considering small field sizes. Larger field sizes increase the complexity of the method.

For method 1, the encoding factors at the source node were chosen randomly from an invertible subset of matrices. From Figure 6.1 and Table 6.1 it can be seen that the RLNC-scheme showed a significant improvement in the probability of a randomly created network to have a higher number of invertible G-matrices using method 1. This improvement is much less pronounced when considering MNC, using method 1.

For method 2, we chose the encoding matrices such that the output of intermediate nodes can be invertible. Method 2 has a much more significant impact on MNC than method 1. This is evident from the increased average seen in Table 6.1. The average decodability of MNC still does not reach that of the unmodified RLNC scheme. However, this does not take into account the additional benefits of MNC in terms of error correction.

When looking only at graphs that show an improvement from the unmodified MNC data using Method 2, as seen in Figure 6.2, we find that the improvement shown in these graphs is even more significant than the unmodified RLNC.

15% of the graphs showed a significant improvement of 50% or more. For these graphs,

the improvement is comparable to RLNC that was enhanced with Method 1. There is possibly an intrinsic characteristic to these graphs that can be identified. If this characteristic could be identified in future work, Method 2 can be used specifically on networks designed according to such graphs.

In Figure 6.4, there is a decline in the decodability of networks as the number of intermediate network nodes increases. This decodability is dependent on the average network connectivity, which decreases with the number of network nodes, assuring minimum network connectivity. The percentage improvement from Methods 1 and 2 stays approximately constant for the different methods and follows a linear pattern. The decodability of MNC Method 2 at thirty intermediate nodes is similar to that of the unmodified MNC at ten intermediate nodes.

Upper triangular matrices as encoding factors

For this experiment, we focused on a method through which to assure invertible matrices as output encoding matrices to intermediate nodes. We, therefore, allowed only upper triangular matrices with 1's on the diagonal. From the previous experiment, we know that in general, assuring invertible output encoding matrices at intermediate nodes increases the probability of invertible G-matrices. In contrast, this specific case always failed. We also see that the output G-matrix would be uninvertible regardless of the network size and the size of the encoding matrices.

From these results, we conclude that using only upper triangular invertible matrices as encoding matrices does not improve invertibility of the G-matrix. Due to the definite invertibility of the G-matrix we can also conclude that using invertible matrices as encoding matrices or assuring the output encoding matrices of intermediate nodes to be invertible, though it does improve the probability of invertibility, does not assure it. We can also theorise that when eliminating choosing only upper triangular invertible matrices, we might be able to improve on the number of invertible G-matrices.

The influence of network depth on error propagation

From the first experiment, we saw that in order to improve the invertibility of the *G*-matrix, we eliminated some encoding factors. In doing so, we also lower the number of possible combinations created through network coding, therefore limiting the number of eventual *G*-matrices. This might influence the network error-correcting abilities of the network. This experiment in Section 6.4 evaluates the influence of network depth on the number of errors propagated through the system.

The experiment evaluates networks of varying depth over two network noise levels, SNR = 8 and 9dB. For both noise levels, the graphs follow the same form, curving up and flattening slightly as the number of nodes increases. From this, we can conclude that the more extensive the network implementing MNC is, the less the increase of error propagation. In other words, more extensive networks slow down error propagation. This could be intrinsic to MNC and can be researched further.

Influence of burst errors

When evaluating the influence of burst errors of different intensities as seen in Section 6.5, we find that for the parameters discussed in Section 5.5, for a low SNR below 6dB, no data was decoded without errors. For burst occurring 10% of the time with intensities, that range from a factor $1\times$ BSC to a factor $1000\times$ BSC, almost all data packets are decoded error-free above SNR 10dB.

We also saw that the higher the burst intensity, the steeper the decline in the probability of packet error, from all packets being filled with errors to almost no errors.

7.3 Revisiting the research objectives

We can now reevaluate the research objectives listed in Section 1.4 to see how and if the objectives were met through the experiments listed above.

The first objective was to evaluate the different ways to improve on the invertibility of the G-matrix for small Galois fields. This investigation was done mainly through the first investigation, where different methods of choosing encoding factors were evaluated and again in the second test that tests a particular case. The first experiment on invertibility improvement also meets the second objective to compare these methods for RLNC and MNC.

The third objective was met, mainly through Method 2 of the first experiment, where we specifically attempted to create invertible encoding factors as outputs to intermediate network nodes. We found that the decodability is, in fact, higher. The test on upper triangular encoding matrices is a more specific case of this test.

The final objectives were to test the effect of network depth and burst errors on error propagation through a network. This was done choosing only invertible encoding matrices, and the results are seen through the final two experiments.

These objectives answered smaller facets of the greater research question.

7.4 Revisiting the research question

The most prominent experiment done in answering the research question was improving the invertibility through different methods of choosing encoding factors of which the results can be seen in Section 6.2. Through this experiment, we found that choosing encoding factors to show invertibility at the start nodes showed some improvement in the number of invertible G-matrices while doing this at intermediate nodes showed considerable improvement.

Since choosing encoding matrices over a small field to produce invertible output encoding matrices at intermediate nodes showed improvement, we did a further investigation by choosing only encoding matrices that are invertible upper triangular matrices. This, however, resulted in no invertible G-matrices, as seen in Section 6.3. The results from this experiment showed that choosing the encoding factors so that the output encoding matrices are invertible, did not guarantee an invertible G-matrix, even though it would make it more likely. Choosing only invertible upper triangular matrices as encoding factors is also not a viable option, and this can, therefore, in future work be eliminated as an option. Experiments can be done to determine whether it would improve invertibility to eliminate invertible upper triangular matrices at only the start or final nodes.

Choosing only invertible matrices as encoding factors limits the total number of outcomes at the destination node, potentially limiting the effectiveness of error correction. The experiments in Section 6.4 and 6.5 show the effects of errors on such a system; how the network depth and burst errors influence the decodability of data.

7.5 Recommendations for future work

The research presented in this dissertation leaves ample opportunity for further research in the field of network coding.

- In future work, it would be interesting to further investigate the influence that choosing only specific encoding matrices has on the propagation of errors through the networks. This will further the research, started in the final two experiments.
- In the experiment on invertibility improvement, Method 2 can, instead of being applied to all nodes, be applied only to the final nodes directly before the destination node. This would lower the computational complexity, but potentially also the decodability.

- It can also be investigated whether eliminating unit upper and lower triangular matrices or other upper and lower triangular matrices as encoding factors would increase invertibility.
- New research is still being done on the invertibility of matrices such as bisymmetric matrices, and these matrices can be tested as encoding matrices.

Bibliography

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul 2000.

[2] Y. Li, Z. Chi, X. Liu, and T. Zhu, "Chiron: Concurrent High Throughput Communication for IoT Devices," in *MobiSys '18*, 2018, pp. 204–216.

[3] S. Zhang, S. C. Liew, and P. P. Lam, "Physical-Layer Network Coding," in *MobiCom '06 Proceedings of the 12th annual international conference on Mobile computing and networking*, Los Angles, CA, USA, 2006, pp. 358–365.

[4] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, Feb 2003.

[5] T. Ho, M. Médard, R. Kötter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, Oct 2006.

[6] R. Kötter and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3579–3591, Aug 2008.

[7] K. T. Kim, C.-S. Hwang, and V. Tarokh, "Network error correction from matrix network coding," in *2011 Information Theory and Applications Workshop*. IEEE, Feb 2011, pp. 1–9.

[8] R. L. Freeman, *Fundamentals of Telecommunications*, 2nd ed., J. G. Proakis, Ed. New Jersey: John Wiley & Sons, Inc., 2005.

[9] R. W. Yeung, *Information Theory and Network Coding*, 1st ed., R. Gallager and J. K. Wolf, Eds. New York: Springer, 2008.

[10] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the Air: Practical Wireless Network Coding," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 497–510, Jun 2008.

[11] R. Kötter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[12] C. Fragouli and E. Soljanin, "Network Coding Fundamentals," in *Foundations and Trends in Networking*. Hanover: NOW Publishers Inc., 2007, vol. 2, pp. 1–133.

[13] ——, "Information flow decomposition for network coding," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 829–848, 2006.

[14] J. Westall and J. Martin, "An Introduction to Galois Fields and Reed-Solomon Coding," School of Computing, Clemson University, Tech. Rep., 2010.

[15] K. Han, M. Huh, K. T. Kim, and K. Jang, "Matrix network coding based multicast scheme over wireless multihop networks," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, Jan 2014, pp. 211–212.

[16] J. Park, T. Kim, W. Lee, D. Byun, and Y. Bae, "Cache aided Matrix Network Coding based multicast scheme over wireless networks," in *2015 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, Jan 2015, pp. 287–288.

[17] J. Park, J. Kim, and J. Park, "Enhanced wireless multicast architecture based on matrix network coding in content-centric networking," in *2016 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, Jan 2016, pp. 31–32.

[18] C. J. Claassen and A. S. J. Helberg, "Error correction performance comparison of intrinsic MNC with concatenated RLNC," in *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, no. 4-7 September, George, South Africa, 2016, pp. 26–31.

[19] J. Heide, M. V. Pedersen, and F. H. Fitzek, "Decoding algorithms for random linear network codes," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6827 LNCS, pp. 129–136, 2011.

[20] C. J. Claassen, *Comparison of error control schemes in Matrix Network Coded networks*, 2017.

[21] P. Erdös and A. Rényi, "On the strength and connectedness of a random graph," *Acta Mathematica Academiae Scientiarum Hungaricae*, vol. 12, pp. 261–267, 1961.

[22] E. N. Gilbert, "Capacity of a Burst-Noise Channel," *Bell System Technical Journal*, vol. 39, no. 5, pp. 1253–1265, 1960.

[23] E. O. Elliott, "Estimates of Error Rates for Codes on Burst-Noise Channels," *Bell System Technical Journal*, vol. 42, no. 5, pp. 1977–1997, 1963.

[24] Y. Yin, R. Pyndiah, and K. Amis, "Performance of random linear network codes concatenated with reed-solomon codes using turbo decoding," *6th International Symposium on Turbo Codes and Iterative Information Processing, ISTC 2010*, pp. 132–136, 2010.

[25] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and Their Applications*, 1st ed. New York: IEEE Press, 1999.

[26] D. Lay, *Linear Algebra and Its Applications*, 4th ed. Pearson, 2014.

[27] J. Hagenauer, L. Papke, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, 1996.

[28] R. M. Pyndiah, "Near-optimum decoding of product codes: block turbo codes," *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1003–1010, 1998.

[29] G. D. Forney and M. D. Trott, "The Dynamics of Group Codes: State Spaces, Trellis Diagrams, and Canonical Encoders," *IEEE Transactions on Information Theory*, vol. 39, no. 5, pp. 1491–1513, 1993.

[30] J. Lodge, R. Young, P. Hoeher, and J. Hagenaueg, "Separable MAP filters for the decoding of product and concatenated codes," in *Proceedings of IEEE International Conference of Communications*, 1993, pp. 1740–1745.

[31] H. Imai, *Essentials of Error-Control Coding Techniques*. San Diego: Academic Press Inc., 1990.

[32] B. E. Rapp, "Special Matrices," in *Microfluidics: Modelling, Mechanics and Mathematics*. Elsevier, 2017, ch. 25, pp. 497–535.

[33] L. Molnár and P. Šemrl, "Some linear preserver problems on upper triangular matrices," *Linear and Multilinear Algebra*, vol. 45, no. 2-3, pp. 189–206, 1998.

[34] The MathWorks Inc., "Biterr," 2019. [Online]. Available: https://www.mathworks.com/help/comm/ref/biterr.html

[35] ——, "BSC," 2019. [Online]. Available: https://www.mathworks.com/help/comm/ref/bsc.html

[36] H. Nguyen and E. Shwedyk, *A First Course in Digital Communication*, 1st ed. New York: Cambridge University Press, 2009.

[37] R. G. Sargent, "Verification and validation of simulation models," in *Proceedings of the 2011 Winter Simulation Conference*, S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, Eds., 2011, pp. 703–715.

Appendix A

Conference Paper

The following conference paper was presented at and appears in the proceedings of the Southern Africa Telecommunication Networks and Applications Conference (SATNAC) from 1-4 September 2019 at Fairmont Zimbali Resort in Ballito, KwaZulu-Natal, South Africa. ISBN:978-0-6398415-0-2 page 114-119.

A Work-In-Progress paper of our work was also accepted for a poster presentation at SATNAC from 2-5 September 2018 at Arabella Hotel & Spa in Hermanus, Western Cape, South Africa.

Invertibility Testing as an Improvement on the Decodability of Matrix Network Coding

Tipharah van Dyk, Albert Helberg, Melvin Ferreira
School of Electrical, Electronic and Computer Engineering
North-West University, Potchefstroom Campus, South Africa
[1]24973742@student.g.nwu.ac.za
[2]albert.helberg@nwu.ac.za
[3]melvin.ferreira@nwu.ac.za

***Abstract*—Network Coding is used to achieve the maximum flow capacity of a network. The most popular Linear Network Coding scheme is Random Linear Network Coding (RLNC), since no knowledge of the network topology is required to implement this method. The greatest weakness of RLNC is that it is vulnerable to errors. Matrix Network Coding (MNC) was created to improve on the network error correction capability of RLNC. MNC, however, has high complexity due to matrix multiplication and the matrix inversion of the Generator-matrix (G). Another problem is that the G-matrix might not be invertible, in which case, the source packets cannot be decoded at the sink node. In order to improve the decoding capability of MNC, we propose two different methods. The first is to create invertible encoding matrices at the network source node, and the second is to improve the probability of creating invertible encoding matrices as the outputs of the internal network nodes. These methods show an improvement on randomly generated networks and the second method shows a proportional improvement of 50% or more on a certain subset of these networks.**

***Index Terms*—Network Coding, Matrix Network Coding, Generator-matrix invertibility**

I. Introduction

In order to keep up with information demands in the modern world, it is important to continually improve information flow through networks. Improving the information flow allows one to deliver as much information, as fast as possible. Traditionally, information packets are created at the source node and routed through the network to a sink node. The internal nodes, therefore, perform no computations on the packets. This method, however, is not optimal and cannot always reach the maximum flow of a network [1].

Using the min-cut, max-flow theorem the maximum flow that can pass through a network can be determined. The minimum cut, *min-cut* = ν, is defined as the minimum cut of network edges so that no flow passes from the source to the sink. The maximum flow, *max-flow* = ν, is the maximum rate of information flow between nodes. The theorem states that the maximum flow that passes from the source to the sink node of the network is equal to the capacity of the minimum cut between those two nodes [2]. Ahlswede, Cai and Li [1] proved that if internal network nodes are allowed to combine their incoming packets, and transmit these combinations rather than routing the packets one at a time, the maximum flow proposed by the min-cut, max-flow theorem, can be achieved. This process of combining packets at internal network nodes is known as network coding.

In 2003, Li, Yeung and Cai [3] defined a sub-field of Network Coding known as Linear Network Coding (LNC). Linear Network Coding is a coding scheme that considers a block of data as a vector over a particular base field. The internal nodes can then perform linear transformations of these vectors before passing them along [3]. It was proven in [4] that only linear operations are needed to combine the network packets over a Galois field of symbols. This was done with a deterministic network, i.e., a network where the underlying network topology is known and taken into consideration when choosing coding coefficients.

Since the introduction of LNC, various different LNC techniques were proposed. The most well known of these is Random Linear Network Coding (RLNC). In [5], Ho et al. proved that the multicast capacity of a network could be achieved by randomly selecting the linear coding coefficients from a large enough field, F_{2^m}. This coding scheme can be used in networks where the underlying topology is unknown or even dynamically changing.

One of the greatest drawbacks of RLNC is that it is very susceptible to packet transmission errors. An error in a single packet, received at a node, is combined with the other packets and will typically result in the entire transmission being discarded [6].

Matrix Network Coding (MNC) was created to improve on the network error correction capabilities of RLNC [7]. MNC was introduced in 2011 by Kim et al [7] as a non-deterministic coding scheme that is in essence quite similar to RLNC. For MNC, matrices of randomly chosen symbols over a Galois field, F_{2^m}, are used as the encoding coefficients. This differs from RLNC where symbols, instead of matrices, are used as encoding factors [7].

Due to the improvement in network error correction capabilities, MNC performs well in multihop, multicast scenarios, as shown in [8]. This idea was further expanded on in [9] and [10] by using enhanced MNC techniques such as cache aided MNC.

Despite the additional network error correction capabilities of MNC, decoding the scheme proves problematic due to the added complexity of matrix encoding factors. In this paper,

we provide a method to improve on the decodability of both RLNC and MNC by addressing the requirement of having an invertible transfer matrix.

In [7], it was stated that the Galois Field size adds to the computational complexity of the network. We will present techniques that improve the decodability specifically for small field sizes since the decodability goes up as the field size increases.

The rest of the paper is organised as follows: In Section II, the RLNC and MNC-encoding schemes are discussed in detail. In Section III, the main research question, that results from the background information, is stated. In Section IV, we discuss the method by which this research question will be answered. In Section V, the implementation of the methodology is discussed and the final results of this simulation is seen in Section VI. Finally, conclusions are drawn from the results and proposals are made for future work in Section VII.

II. Background

A. Related Work

MNC was introduced by Kim et al. in [7], who proposed that encoding factors at internal network nodes are chosen such that the output encoding factors of those nodes would be invertible. The paper however only served as a conceptual introduction to the routing scheme and minimally small networks were used.

In 2016, Claassen and Helberg [11] did simulations to determine the effect of field size on the invertibility of the G-matrix . It was found that due to the increased size of the encoding matrices, the field size of MNC can be smaller than that of RLNC. They also proposed that invertibility of the G-matrix could be improved by choosing the correct encoding factors.

B. Random Linear Network Coding Scheme

In networks where network coding has been implemented, all the incoming packets are combined to form linear combinations that can then be passed into the network. At each internal network node where more than one packet is received, these packets are also linearly combined. The linear combinations are created using randomly selected coding coefficients from a Galois field F_{2^m}. When enough linearly independent packets are received at the sink node, they can be decoded. The full and more detailed description of RLNC that follows below is based on the work of Ho et al. [5].

1) Encoding RLNC network packets: We will consider a network with N source packets $\mathbf{P} = [P_1, P_2, ..., P_N]$, each of length L defined over a Galois field F_{2^m}. The network must have a *min-cut* $\geq N$ to avoid capacity constraints. The output packet P_S^t for each of the outgoing edges, t, of the source node, S, can be calculated as follows:

$$P_S^t = \sum_{i=1}^{N} \alpha_i^t P_i. \tag{1}$$

In (1) random encoding factors, $A_S^t = [\alpha_1^t, \alpha_2^t, ..., \alpha_N^t]$ are chosen for each outgoing edge. These encoding factors are sent, along with the newly encoded data, as header information. The header information increases the overhead of each packet, but is outweighed by the benefits of RLNC.

The process described above is similar to the encoding process at internal network nodes. At an internal node, all h incoming packets are combined. h is not necessarily equal to N. If only one packet is received, that packet is multicast, as is, over all of the node's outgoing edges.

If h packets are received at internal node, T, a new set of encoding coefficients, $\mathbf{A_T} = [\alpha_1, \alpha_2, ..., \alpha_h]$ are randomly chosen from the finite field. The incoming packets, $P_{T_{in}}$, are then encoded similar to (1) as seen in (2):

$$P_{T_{out}} = \sum_{i=1}^{h} \alpha_i^t P_{T_{in}}. \tag{2}$$

The encoding vectors from each incoming packet, r, are constructed into vectors as seen in (3):

$$\mathbf{g}^r = [\alpha_1^r, \alpha_2^r, ..., \alpha_N^r] = [g_1^r, g_2^r, ..., g_N^r] \tag{3}$$

with $r = 1 \rightarrow h$.

By combining all the $\mathbf{g}^r$ vectors with $\mathbf{A_T}$ we are able to create a new encoding vector that is multicast as the header information from node T. This combination is done as seen in (4):

$$g_i = \sum_{r=1}^{h} \alpha_i g_i^r. \tag{4}$$

2) Decoding RLNC network packets: At the sink node, R, the incoming packets have to be decoded to the original input packets $\mathbf{P}$. N linear independent combinations of the source packets are needed for effective decoding.

The encoding vectors of each of the received packets are combined into the generator matrix, G, as in (5):

$$G = [\mathbf{g}^1, \mathbf{g}^2, ..., \mathbf{g}^N] = \begin{bmatrix} \alpha_1^1 & \alpha_1^2 & \cdots & \alpha_1^N \\ \alpha_2^1 & \alpha_2^2 & \cdots & \alpha_2^N \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_N^1 & \alpha_N^2 & \cdots & \alpha_N^N \end{bmatrix}. \tag{5}$$

The generator matrix is used to decode each of the input packets as seen in (6):

$$P(j) = P_R(j).G^{-1} \tag{6}$$

for $j = 1 \rightarrow L$ denoting the j-th element in the source packet.

The probability of invertibility of this G-matrix increases exponentially as the Galois field size increases. In this paper we investigate the decodability of RLNC for smaller field sizes and propose a method for improvement.

C. Matrix Network Coding Scheme

Matrix network coding, as described in [7] is an expansion on RLNC. We will describe it below in contrast to RLNC as previously explained.

1) Encoding MNC network packets: The encoding of MNC differs from RLNC in that the source packets, defined over a Galois field F_{2^m}, are constructed into $p \times q$ matrices where the original packet length is $L = pq$. These matrices can be constructed by representing the input data, P_i, as q, p-dimensional vectors using (7)

$$P_i = (\mathbf{P}_i(1), \mathbf{P}_i(2), ..., \mathbf{P}_i(q)) \tag{7}$$

with

$$P_i(j) = (P_i((j-1)p+1), P_i((j-1)p+2), ..., P_i(jp)) \tag{8}$$

for all $j = 1 \rightarrow q$.

Encoding is quite similar for MNC and RLNC. The main difference is that instead of the encoding factors $\mathbf{A_T}$, used for RLNC, MNC uses matrices. For example, if k packets are received at the internal node T, k $p \times p$ matrices, $(A_1, A_2, ..., A_k)$, are chosen over the field F_{2^m} to encode these packets.

These matrix combination coefficients are used to calculate the output matrix of the internal node as in (9):

$$P = \mathbf{P}(1), \mathbf{P}(2), ..., \mathbf{P}(q) \tag{9}$$

where for $j = 1 \rightarrow q$

$$\mathbf{P}(j) = \sum_{i=1}^{N} A_i \cdot P_i(j). \tag{10}$$

From the differences mentioned between RLNC and MNC thus far, it should be noted that MNC reduces to RLNC if $p = 1$. The encoding matrices will reduce to encoding coefficients, and L will be left as $L = q$. For this reason, in this paper, the term MNC implies $p \geq 2$.

2) Decoding MNC network packets: At the sink node, different matrix combined packets will be received that are decoded using the matrix encoding coefficients found in each packet header. The encoding coefficients are combined into a matrix known as the G-matrix seen in (11):

$$G = \begin{bmatrix} A_1^1 & A_1^2 & \cdots & A_1^N \\ A_2^1 & A_2^2 & \cdots & A_2^N \\ \vdots & \vdots & \ddots & \vdots \\ A_N^1 & A_N^2 & \cdots & A_N^N \end{bmatrix}. \tag{11}$$

The original packets, $\mathbf{P}_i$, are recovered similar to Equation (6) as seen in (12):

$$P(j) = P_o(j).G^{-1} \tag{12}$$

for $j = 1, 2, ..., q$.

For both RLNC and MNC, the input packets can only be recovered if the G-matrix is invertible.

Decoding is done with the assumption that no errors occurred during transmission.

The greatest drawbacks of MNC can be seen from (12). The complexity of decoding can be troublesome with larger G-matrices. According to [7], if the header overhead of MNC and RLNC is kept the same, the complexity of MNC is estimated as $\mathcal{O}(N^3L^3m^2)$ while RLNC in comparison has a complexity of $\mathcal{O}(N^3L^4m^2)$ when N source packets are combined, at each node, with an $L \times L$ matrix chosen over a finite field F_{2^m}. The final drawback of both RLNC and MNC, is that the G-matrix must be invertible, which is often not the case for small field sizes.

III. Research Question

Since encoding factors can be chosen at random at the different nodes in the network, we have discretion in choosing these encoding factors in such a manner that the end result might be more easily decodable. This was exposed in [7] and [11] as a method to improve MNC.

In this paper, we investigate different ways of improving the invertibility of the G-matrix to improve the decoding probability of both MNC and RLNC for small Galois fields.

IV. Methodology

To address the research question, we created flow network graphs through which various different scenarios could be evaluated by means of Monte Carlo simulations.

A. Network Setup

A network can be represented as a graph, $G = (V, E)$. where V is the set of vertices or nodes and E the set of edges, representing the communication links, that connect the nodes.

A directed, acyclic graph is a graph where all the edges are oriented and creates no directed cycles, such that there is a sequence of edges and no path can be found from any node back to itself [12].

Érdos-Rényi graphs were used in [13] and [11] to assure that the underlying network structure has as little influence on the outcome of the simulations as possible when considering randomised networks with unpredictable topological structures and changes. This will allow for other factors, such as the encoding schemes, to be changed and evaluated. Since Érdos-Rényi graphs are especially suited when trying to randomise the graphs simulated, they closely approximate the behavior of practical networks.

For an Érdos-Rényi graph, each of the network nodes has a chance to be connected to any of the other nodes in the network by a probability δ. These connections are created independent of one another and the distance between the nodes, therefore, creating randomly connected graphs. Since the connections are made randomly, graphs can be either sparsely or densely connected.

According to [12], a graph is optimally connected if the minimum number of outgoing edges, d_{min}, node connectivity, κ, and edge connectivity, λ, are the same. κ refers to the minimum number of nodes to remove before a graph becomes disjoint while λ is the minimum number of edges.

From [12] the probability, δ for a network with n nodes was chosen as shown in (13):

$$\delta = \sqrt{\frac{n}{n-1}}. \tag{13}$$

The random networks that result from Érdos-Rényi connections can have multiple source and sink nodes. We, however, only want to simplify our networks to networks with one source and sink node. The Érdos-Rényi graph is therefore only used to represent the interior of the graph. A source and sink node are then added and connected to all the potential source and sink nodes of the internal network, respectively.

If there are more connections at the source node than at the sink node the direction of the graph can be reversed to assure that the *min-cut, max-flow* theorem is realised. An example of a resulting graph can be seen in Fig. 1.

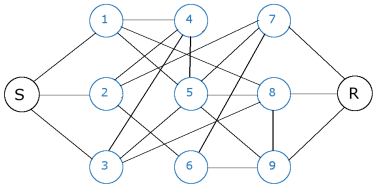

Fig. 1. Example of Érdos-Rényi based random network graph with added source (S) and sink (R) nodes

For simplicity the capacity of each of the edges is set to 1 and it is assumed that the distance between nodes is irrelevant. It is also assumed that any packets that arrive at a network node arrive simultaneously to create the greatest possibility for network coding.

V. Implementation

Two methods were modeled to determine the effect of strategically choosing the encoding factors for both RLNC and MNC.

The parameters chosen to realise the methods are listed in Table I.

TABLE I
Simulation Parameters

Parameter	Value
Internal network nodes	10:5:30
Number of graphs of each node count	1000
Iterations per graph	100
Field size (F_{2^m})	2
MNC p-value	2

The unmodified, RLNC- and MNC-schemes where used as the benchmark cases to compare to the two methods listed below:

1) Method 1: For the first method the source encoding factors are compiled into a matrix as they would be in Equation (5) or (11) to create a G-matrix. This matrix was tested and recreated randomly until it was found to be invertible. This method increases the computational load at the source node only.

2) Method 2: For method 2, it was required that the encoding factors, that are the output of any internal network node, should be invertible.

This was achieved by first choosing all encoding factors, whether at the start or internal nodes, to be only invertible matrices. For each of the internal encoding factors to be chosen as invertible matrices, these factors should be of a size $p \geq 2$. This stipulation limits this method to apply only to MNC.

Computations were, performed at the internal node to determine the output matrix. If the output matrices are not invertible, up to 10 iterations where done for which most cases resulted in an invertible matrix.

This method increases the computational load at all of the network nodes.

VI. Simulation Results

By applying the methodology and implementation, as described in Sections IV and V, the following results were found.

Fig. 2 shows a Complementary Cumulative Distribution Function (CCDF) of the percentage invertible G-matrices received at the sink node of randomly created graphs. From Fig. 2 we see that 80% of random networks, that implement the unmodified RLNC-scheme, have only a 40% probability of correctly decoding received packets at the sink node. Similarly, there is an increased probability from 55% to 90% that a random network that implements RLNC with method 1 will have a 50% probability of correct decoding.

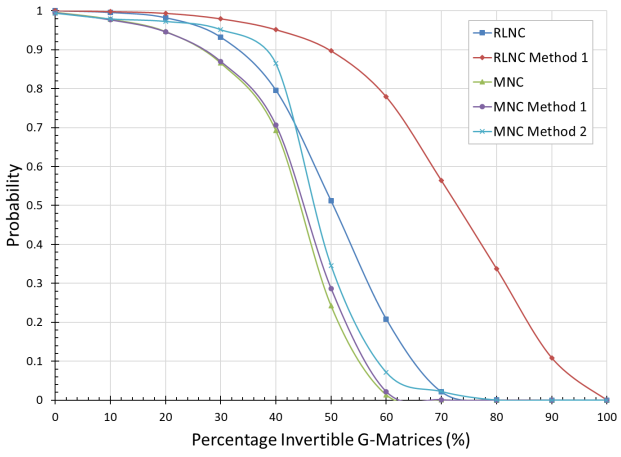

Fig. 2. Probability of a randomly generated network graph to have a certain percentage of invertible G-matrices, expressed as a CCDF

The average, median and standard deviation of each method in Fig. 2 can be seen summarised in Table II:

TABLE II
STATISTICAL PARAMETERS FOR THE DECODABILITY OF RANDOMLY GENERATED NETWORK GRAPHS

	Average	Median	Standard Deviation
RLNC	50.0	51	12.2
RLNC Method 1	71.5	73	16.4
MNC	42.8	45	11.5
MNC Method 1	43.5	46	11.8
MNC Method 2	47.5	48	10.8

Consider the graphs for MNC in Fig. 2. It is clear that both methods 1 and 2 show some improvement in decodability over the unimproved case, although not as much as with RLNC. However, not all graphs showed an improvement, with only 65% of randomly selected networks showing some improvement. Closer investigation of these random networks that show improvement is shown in Fig. 3.

From Fig. 3, it is seen that both methods 1 and 2 improve the decodability of MNC. In the case of method 2, the improvement shows that on average, in 55% of the cases that MNC has a better probability of successful decoding than the baseline RLNC.

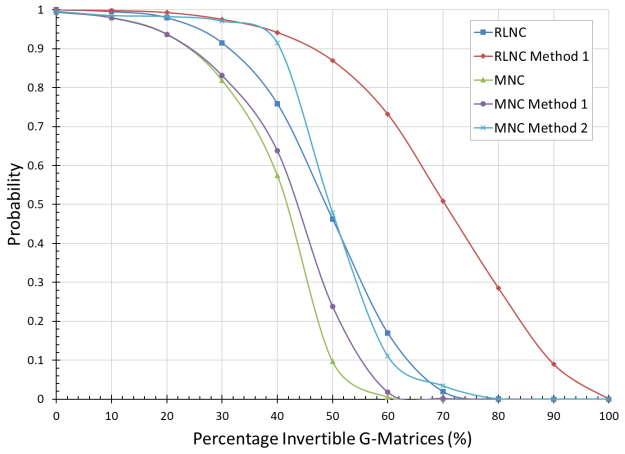

Fig. 3. Probability of a subset of network graphs, where method 2 shows improvement, to have a certain percentage of invertible G-matrices, expressed as CCDF

Fig. 4 shows only the 15% of random networks that have more than 50% improvement in decoding probability.

From Fig. 4, it is seen that in some of the cases that have more than 50% improvement from their baseline implementations, MNC with method 2 even improves on RLNC with method 2.

The statistics from Fig. 4 can be seen in Table III.

It can be noted that the average for MNC method 2 improves and all the other values are lower than in Table II. From this we can see that method 2 works especially well for graphs that would otherwise have a lower than average number of invertible G-matrices.

Fig. 5 shows how the number of invertible G-matrices, and, therefore, the decodability of the network changes for different numbers of internal network nodes.

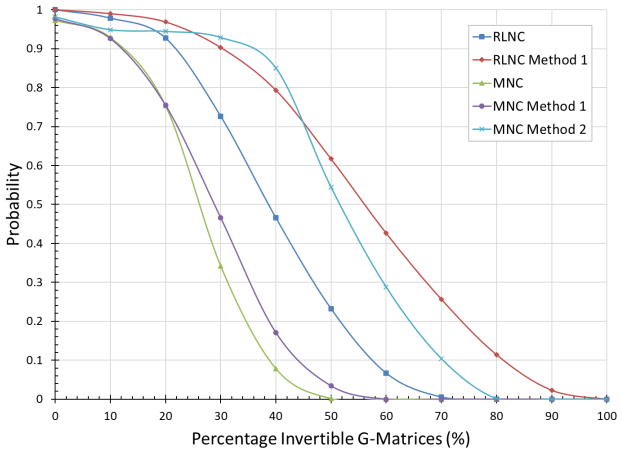

Fig. 4. Probability of a subset of network graphs to have a certain percentage of invertible G-matrices, where method 2 shows an improvement of 50% or more, expressed as CCDF

TABLE III
STATISTICAL INFORMATION DERIVED FROM FIG. 4 TO DESCRIBE THE DIFFERENT METHODS ANALYSED

	Average	Median	Standard Deviation
RLNC	39.6	40	13.9
RLNC Method 1	56.4	57	19.2
MNC	26.2	27	10.4
MNC Method 1	28.8	30	12.3
MNC Method 2	51.6	51	16.2

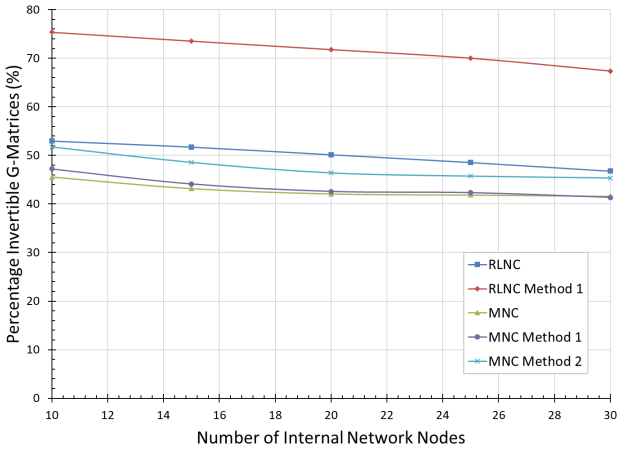

Fig. 5. Percentage invertible G-matrices for graphs with various numbers of internal network nodes

VII. CONCLUSION

Two different methods were tested to evaluate their impact on increasing the likelihood of invertibility of the G-matrix for randomly created networks when considering small field sizes.

For method 1, the encoding factors at the source node were chosen randomly from an invertible subset of matrices. From Fig. 2 and Table II it can be seen that the RLNC-scheme showed a large improvement in the probability of a randomly

created network to have a higher number of invertible G-matrices using method 1. This improvement is much less pronounced when considering MNC, using method 1.

For method 2, we chose the encoding matrices such that the output of internal nodes can be invertible. Method 2 has a much larger impact on MNC than method 1. This is evident from the increased average seen in Table II. The average decodability of MNC still does not reach that of the unmodified RLNC scheme. However, this does not take into account the additional benefits of MNC in terms of error correction.

When looking only at graphs that show an improvement from the unmodified MNC data using method 2, as seen in Fig. 3, we find that the improvement shown in these graphs is even greater than the unmodified RLNC.

15% of the graphs showed a significant improvement of 50% or more. For these graphs the improvement is comparable to RLNC that was enhanced with method 1. There is possibly an intrinsic characteristic to these graphs that can be identified. If this characteristic could be identified in future work, method 2 can be used specifically on networks designed according to such graphs.

In Fig. 5 there is a decline in the decodability of networks as the number of internal network nodes increases. This decodability is dependent on the average network connectivity which decreases with the number of network nodes, assuring a minimum network connectivity. The percentage improvement from methods 1 and 2 stays approximately constant for the different methods and follows a linear pattern. The decodability of MNC method 2 at 30 internal nodes is similar to that of the unmodified MNC at 10 internal nodes.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the financial support of this study by the Telkom CoE at the NWU and the National Research Foundation under grant nr TP14081892668.

REFERENCES

[1] R. Ahlswede, Ning Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, jul 2000.

[2] R. W. Yeung, *Information Theory and Network Coding*, 1st ed., R. Gallager and J. K. Wolf, Eds. New York: Springer, 2008.

[3] S.-Y. Li, R. Yeung, and Ning Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, feb 2003.

[4] R. Kötter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, oct 2003.

[5] T. Ho, M. Medard, R. Kötter, D. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, oct 2006.

[6] R. Kötter and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3579–3591, aug 2008.

[7] K. T. Kim, C.-S. Hwang, and V. Tarokh, "Network error correction from matrix network coding," in *2011 Information Theory and Applications Workshop*. IEEE, feb 2011, pp. 1–9.

[8] K. Han, M. Huh, K. T. Kim, and K. Jang, "Matrix network coding based multicast scheme over wireless multihop networks," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, jan 2014, pp. 211–212.

[9] J. Park, T. Kim, W. Lee, D. Byun, and Y. Bae, "Cache Aided Matrix Network Coding Based Multicast Scheme Over Wireless Networks," *2015 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 287–288, 2015.

[10] J. Park, J. Kim, and J. Park, "Enhanced wireless multicast architecture based on matrix network coding in content-centric networking," in *2016 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, jan 2016, pp. 31–32.

[11] C. J. Claassen and A. S. Helberg, "Error correction performance comparison of intrinsic MNC with concatenated RLNC," in *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, no. 4-7 September, George, South Africa, 2016, pp. 26–31.

[12] A. H. Dekker and B. D. Colbert, "Network robustness and graph topology," pp. 359–368, 2004.

[13] S. von Solms and A. Helberg, "Modified Earliest Decoding in Networks that Implement Random Linear Network Coding," *SAIEE Africa Research Journal*, vol. 103, no. 4, pp. 165–171, dec 2012.

Tipharah G. van Dyk obtained her Bachelors in Computer and Electronic Engineering in 2017 at the North-West University, Potchefstroom. She further pursued her studies in 2018 by registering for a Master's degree in Computer and Electronic Engineering with a focus on Telecommunications and Network Coding in collaboration with the Telkom Center of Excellence.