View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UPCommons. Portal del coneixement obert de la UPC

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Bit Error Combating Network Coding Techniques

Pau Bernat Guri Hundertmark

Advisors:
Dr. Nikolaos Thomos
Prof. Pascal Frossard

June 2010

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Contents

List of figures

Chapter 1

Introduction

1.1 Motivation

In the latter years multimedia streaming has grown rapidly. Applications such as youtube, for example, are used worldwide. This use of multimedia streaming applications worldwide needs huge amounts of network resources. In order to avoid overloading these network resources some solutions have been proposed. One of them is Network Coding. With Network Coding the amount of network resources employed to send some information between network nodes can be reduced. This happens because with network coding we can reduce the power consumption and also we can minimize the delay. However, the use of network coding leads to another problem. It is that with network coding an error in a received packet is propagated to all others packets. The reason why an error is propagated to all other packets is because network coding consists in combining different packets. Thus, if a packet is erroneous when combined with others, they will be erroneous too. These erroneous packets will be combined in other nodes with some other packets, and these last ones will be erroneous too. This means that we use less network resources but losing quality. To avoid this, packets have to be protected with some error protecting codes for the receiver to be able to recover the packet sent. But combining network coding with error protecting codes is not trivial because this leads to an error propagation problem.

Multimedia streaming is becoming more important in personal communications. Particularly, mobile communications. Nowadays 3G phones are available to everybody. Applications such TV on the phone are growing rapidly. Not only these applications are becoming important but high definition demand is also rising. This also leads to a big demand for network resources, and in wireless networks the resources are even more valuable than in wired networks as the spectrum is limited.

1.2 Related work

Several solutions have been proposed to use network coding with error protecting codes. Some theoretical works of Kötter and Kschischang [13] assume a non coherent network coding model, where neither the transmitter nor the receiver have knowledge of the channel transfer characteristics and they propose an encoding and decoding algorithm for Gabidulin codes. An improvement of the decoding algorithm is proposed in [17], there they propose using Fourier transform method to decode faster Gabidulin codes. Following these works, [14] shows that network error correcting codes achieving the Singleton bound can be easily constructed for both coherent and non coherent network coding. Another work by Danilo Silva and Frank R. Kschischang [15] explains how to use erasures and deviations in the decoding of network coding error protecting codes. A feasible decoder architecture for the codes proposed by Kötter and Kschischang is shown in reference [16]. Also, a practical solution for using network coding in wireless mesh networks is proposed in [18] where they propose the COPE [18] system which bridges theory with practice. An improvement on this solution is MIT's MIXIT [1] which is one of the best and most complex. This system gives a good performance but the complexity is also high, as it needs knowledge of the network, and bidirectional communication between nodes. The main idea is than a node transmits only that symbols that have no errors. Also, the receiver can ask for retransmission to the intermediate nodes if it notices some symbols is missing or wrong. MIXIT gives better performance than our system but it is more complex as well. Our system could be combined with MIXIT and the resulting system would be better than both on its own. For more information about MIXIT system the reader can refer to [1]. Another solution that has been proposed is a decoder for Reed Solomon product codes that uses a MAP decoder. These kinds of codes are called Distributed Product Codes, and more information can be found at reference [2].

1.3 Goal

The aim of this project is to propose a system that with less complexity than MIXIT could give better performance than Distributed Product Codes in media streaming over wireless networks.

To do that we try with different kinds of error protecting codes, we improve the product codes proposed on [2] and we combine these with some traditional channel coding techniques that.

We see that we can get an amount of correctly received packets high enough for multimedia streaming in wireless network for a minimum SNR. There are some drawbacks though. We decode in every node of the network which adds computational cost, and also we can't get a 100% of correct packets when there are no errors in the network.

1.4 Overview

This report is organized as follows. Chapter 2 contains the theoretical information needed by the reader to understand the methods used in the implementations, and chapter 3 contains the practical implementations we tested in the project.

Chapter 2

Theory

2.1 Network coding

The main idea of network coding is quite simple. Every intermediate node in the network combines the packets it has received and sends the combined packets to the next hops. Following this approach there is a gain in the throughput and the delay is reduced. Let's clarify this idea with a simple example shown in figure 1.

Just imagine that server S1 wants to send a packet X1 to the client C2, and server S2 wants to send another packet X2 to client C1. With the traditional way S1 send the packet X1 to the relay R1, and it send the packet to the relay R2, and this last one send it to the client C2. The same procedure is followed for packet X2. The thing is that both packets cannot be sent at the same time when the communication is limited to one packet per time slot. Instead, if we allow relays to combine packets, we can use the procedure that follows. S1 broadcasts packet X1 through all output links, so relay R1 and client C1 get the packet X1. Server S2 does the same so R1 gets X2 and client C2 gets it also. Now R1 combines both packets that it has received from its input links, resulting X1+X2. R1 sends X1+X2 to relay R2 and R2 broadcasts it. So, client C1 has packets X1 and X1+X2, and client C2 has X2 and X1+X2. Now for client C1 to recover packet X2 he only has to combine the packets it has, i.e. X1+(X1+X2) = X2. The same procedure is followed for client C2.

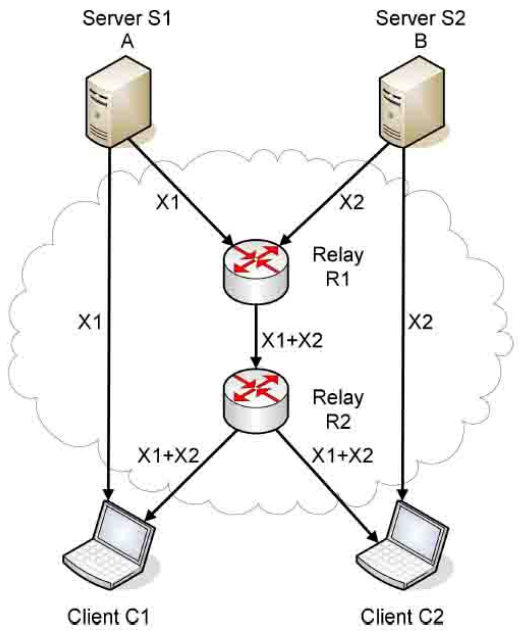

Figure 1. Toy example of a simple multicast transmission. If R1 is allowed to combine packets, throughput is improved and delay is reduced.

With this simple example we have proven that client C1 can get X2 and client C2 can get X1 at the same time that with the traditional way only one of the destination nodes can get the desired packet. Therefore, throughput has been increased and delay has been reduced.

We will next show how network coding can bring some advantages when it is applied to wireless networks.

To demonstrate how network coding works over wireless network we use the example shown in figure 2.

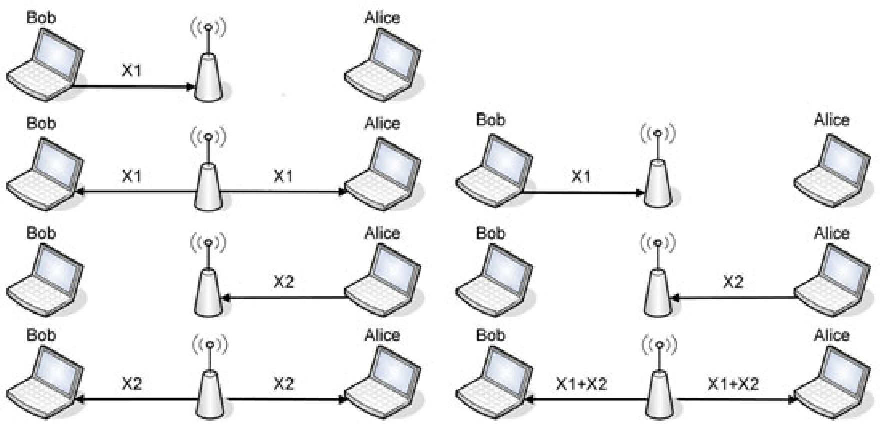

Figure 2. Example of wireless communication comparing the same system using network coding and store-and-forward approach. Bob and Alice exchange a pair of packets using 3 transmissions instead of 4.

Consider that Bob wants to send packet X1 to Alice, and Alice wants to send packet X2 to Bob. Also consider that direct communication between Bob and Alice is not possible because transmission range is limited, so a client cannot reach directly the other. By the traditional way each client must first send his packet to the antenna and then the antenna will send the packet to the other packet. Therefore, the communication takes four time slots. However, if we enable the antenna to use network coding, the communication between both clients requires three time slots. Bob sends his packet to the antenna. Then Alice sends her packet to the antenna. The antenna combines both packets and sends the combinations to both receivers. The delay, therefore, has been reduced.

Above, we have explained the strictly necessary things about network coding for the reader to understand our system. For those readers who want more information about network coding refer them to [3]. For even further information about network coding please refer to [4].

2.2 Error correcting codes

In communications, some errors may occur in the channel. That means that some bits sent by the sender can be flipped because of the noise generated at the channel. When this happens the receiver gets a stream of bits that is not the same to the one the source sent. In multimedia streaming this may result in loss of quality.

There are some ways of preventing quality degradation from happening. The most common way is Forward Error Correction [4,5] (FEC). The main idea is that the sender adds redundant data to the transmitted messages. This allows the receiver to detect and eventually correct (within some bound) some errors that have been generated by the channel.

Maybe the most popular FEC code is Reed-Solomon [4,5]. Next, we explain briefly how it works. We also explain Maximum Rank Distance (MRD) codes [20], which can improve performance [8].

2.2.1 Reed Solomon Codes

In coding theory, Reed–Solomon (RS) codes are linear error-correcting codes proposed by Irving S. Reed and Gustave Solomon. They presented a systematic way of building codes that could detect and correct multiple symbol errors. By adding t redundant symbols to the data, an RS code can detect any combination of up to t erroneous symbols, and correct up to $t/2$ symbols. The choice of t depends on the designer of the code, and may be selected within wide limits.

Reed-Solomon codes are used in important applications from deep-space communication to consumer electronics [19]. They are used in consumer electronics such as CDs, DVDs, Blu-ray Discs, in data transmission technologies such as DSL, and also in digital TV broadcasting.

The Reed-Solomon code is a $[n, k, n\text{-}k+1]$ code; in other words, it is a linear block code of length n (over a Galois field F) with dimension k and minimum Hamming distance $n\text{-}k+1$. The error-correcting ability of a Reed–Solomon code is determined by its minimum distance, or equivalently, by the number of redundant symbols $n\text{-}k$. A Reed–Solomon code can correct up to $(n - k) / 2$ erroneous symbols, i.e., it can correct half as many errors as there are redundant symbols added to the block. If the locations of the error symbols are not known in advance, then a Reed–Solomon code can correct up to $(n - k) / 2$ erroneous symbols, i.e., it can correct half as many errors as there are redundant symbols added to the block. Sometimes error locations are known in advance (e.g., “side information” in demodulator signal-to-noise ratios) these are called erasures. A Reed–Solomon code is able to correct twice as many erasures as errors, and any combination of errors and erasures can be corrected as long as the relation $2E + S \leq n - k$ is satisfied, where E is the number of errors and S is the number of erasures in the block.

For practical uses of Reed–Solomon codes, it is common to use a finite field F with 2^m elements, $GF(2^m)$. In this case, each symbol can be represented as an m-bit value.

For further information about information theory and FEC codes we refer the readers to check [4,5].

2.2.2 Maximum Rank Distance Codes

While RS codes are based in Hamming distance (number of different bits between two words) MRD codes are based in Rank distance which is defined as: $d_r(A,B) = rank(A-B)$, where A and B are MRD coded-words. This means that maximum rank distance codes can correct rank errors, i.e. they can correct an error whose rank is less or equal to the correction capacity of the code. The rank of an error word is the number of independent erroneous symbols. As an example:

Imagine that the error word is $\begin{pmatrix} 3 & 3 & 0 \end{pmatrix}$, we can express it in a binary base: $\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$, the zero word doesn't count to compute the rank of a matrix, and the element $\begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$ appears twice in the matrix, thus the only independent element is $\begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$. Therefore the rank of this error word is 1.

The main property of MRD codes is that between any two code words there is a maximum rank between all of them.

MRD codes are also linear *[n,k,n-k+1]* codes, and are characterized by:

- m, size of the Galois Field where the code is defined.
- n, total number of symbols in a code word
- k, amount of information symbols in a code word
- d_r, rank distance, which in MRD codes is: $d_r = n - k + 1$

The correction capacity of MRD codes is defined similar to Reed-Solomon codes correction capacity, but using rank distance instead of hamming distance: $c = \left\lfloor \frac{d_r - 1}{2} \right\rfloor$.

More information about rank metrics and maximum rank distance codes can be found in [7, 8].

A particular case of MRD codes are Gabidulin codes.

2.2.2.1 Gabidulin codes

Gabidulin codes, proposed by Ernst Gabidulin [20], are a particular case of MRD codes with the constraint $n \le m$. Thus, if the code is defined over the Galois Field $GF(2^3)$, the maximum codeword size is $n = 3$.

Any linear block code is characterized by the parity check matrix *H*, and the generator matrix *G*. Next, we explain how to build these matrices and the encoding and decoding procedures of Gabidulin codes.

Encoding

Usually in FEC codes the generator matrix can be found without knowledge of parity check matrix, on contrary Gabidulin codes the parity check matrix has to be found before obtaining the generator matrix. Parity check matrix has the following form:

$$H = \begin{bmatrix} h_1 & h_2 & \dots & h_n \\ h_1^{[1]} & h_2^{[1]} & \dots & h_n^{[1]} \\ h_1^{[2]} & h_2^{[2]} & \dots & h_n^{[2]} \\ \dots & \dots & \dots & \dots \\ h_1^{[d-2]} & h_2^{[d-2]} & \dots & h_n^{[d-2]} \end{bmatrix}$$

where h_i are linearly independent element of the Galois field where the code is defined, d is the rank distance, and $h_j^{[i]} = (h_j)^{2^i}$.

Once we have found the parity check matrix of the code we can find the generator matrix that fulfills these two constraints:

$$G \cdot H^T = 0 \qquad G = \begin{bmatrix} g_1 & g_2 & \dots & g_n \\ g_1^{[1]} & g_2^{[1]} & \dots & g_n^{[1]} \\ g_1^{[2]} & g_2^{[2]} & \dots & g_n^{[2]} \\ \dots & \dots & \dots & \dots \\ g_1^{[d-2]} & g_2^{[d-2]} & \dots & g_n^{[d-2]} \end{bmatrix}$$

where g_i are linearly independent elements of the Galois field, and H^T represents the transposed matrix of H.

The procedure to find a G that satisfies these properties is not trivial, however a procedure to find it is proposed at [6].

Decoding

The decoding of Gabidulin codes is similar to that of the Reed Solomon codes. Therefore the decoding is described by following this flowchart:

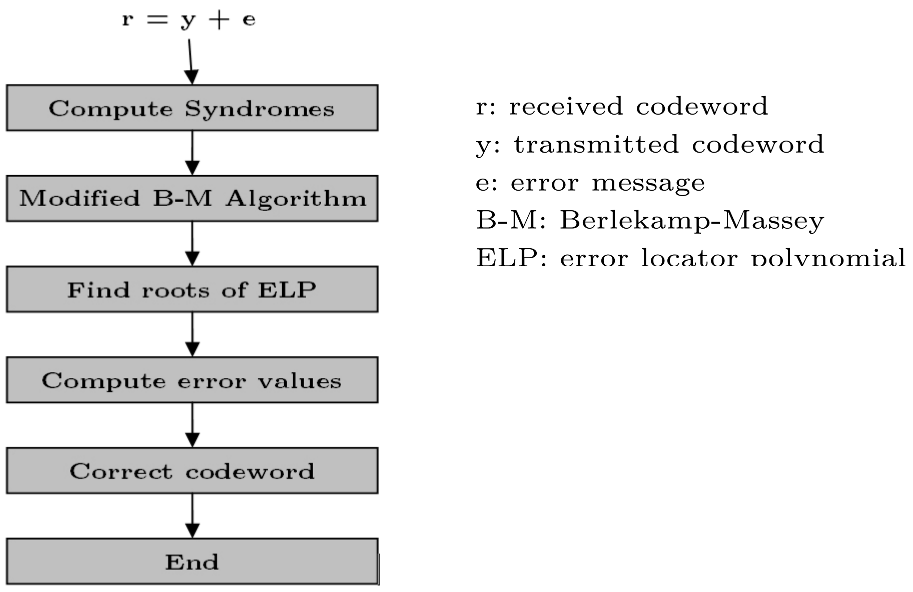

Figure 3. Flowchart of the decoding algorithm of Gabidulin codes.

- Compute the syndromes: it consists in multiplying the received word with the parity check matrix. The resulting vector is the vector of syndromes.

- Modified Berlekamp-Massey Algorithm: this algorithm allows us to find the error locator polynomial, whose roots are the error positions. The algorithm is quite complex, but an extended explanation about it can be found at [9].

- Find the roots of the ELP: ELP is the error locator polynomial. Its roots contain information about the error positions.

- Compute error values: the error values are found solving a linear system of equations involving the roots of the ELP and the syndromes.

- Correct the received word: once the position of the errors and their values are known the correction of the received word is trivial.

For more detail regarding the decoding Gabidulin codes the interested readers are referred to [9].

As MRD codes allow us to correct rank errors we adopt them in our systems.

In this report, we refer to a Gabidulin code with m, n and k (where m is the size of the Galois field *(GF(2^m))* where the code is defined, n is the size of a codeword and k is the number of information symbols each codeword contains), as *MRD(m, n, k)*.

2.3 Combining Gabidulin codes with NC

In order the receiver to be able network decode, a unique header has to be added to the words that need to be sent. We use a vector which contains a one and all others zero as a unique header, thus the operations done by the network relays will affect this matrix, allowing the receiver to decode them.

To clarify the idea we show an example where three code words will be sent:

If we want to send three MRD(3,3,1) code words: X_1, X_2, X_3, *and each* X_i *equals to* $X_i = (x_{i1} \quad x_{i2} \quad x_{i3})$, *we have to add as a unique header the identity matrix 3x3. So the resulting message will be like this:* $Z = \begin{bmatrix} 1 & 0 & 0 & X_1 \\ 0 & 1 & 0 & X_2 \\ 0 & 0 & 1 & X_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_{11} & x_{12} & x_{13} \\ 0 & 1 & 0 & x_{21} & x_{22} & x_{23} \\ 0 & 0 & 1 & x_{31} & x_{32} & x_{33} \end{bmatrix}$

Then, considering that the combinations done by the network can be modeled by a matrix NC and a simple matrix multiplication, we can observe how these combinations are recorded in the unique header.

$$NC = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow NC \cdot Z = \begin{bmatrix} a_{11} & a_{12} & a_{13} & Y_1 \\ a_{21} & a_{22} & a_{23} & Y_2 \\ a_{31} & a_{32} & a_{33} & Y_3 \end{bmatrix}$$

Where Y_i are linear combinations of X_i.

The problem of this procedure is that when an error occurs in the coefficients sub-matrix of our code then the whole word is erroneous. This happens because an error in matrix NC causes the destination to receive a combination that doesn't match with the actual combination done by the network. As MRD codes are linear codes, i.e. any combination of MRD words is another MRD word, the receiver is unable to recover the original words because, since the words it has received are valid MRD coded-words, it does not classify them as errors. Somehow we have to protect the matrix *NC*. However, this matrix is created in the network so the sender doesn't know it. Then how can we protect this matrix? A solution is proposed below.

2.3.1 Systematic Gabidulin codes

A systematic code means that the information symbols that have to be encoded remain unchanged after encoding. Let's show it:

If we want to encode one symbol x_1 with a non systematic MRD(3,3,1) code we will get as encoded word $\left(\hat{x}_{11} \quad \hat{x}_{12} \quad \hat{x}_{13}\right)$ where $\hat{x}_{11} \neq x_1$. If we encode with a systematic MRD(3,3,1) code instead, we will get $\left(x_{11} \quad x_{12} \quad x_{13}\right)$ where $x_{11} = x_1$.

The reason why we are looking for a systematic encoding of Gabidulin codes is because it allows us to encode the header and still have the identity matrix. Therefore it will be able to decode errors in the header matrix without requiring extra computational cost on our system.

A systematic way of constructing systematic Gabidulin codes has not been proposed, to the best of our knowledge. Here, we propose an easy and simple way of doing that.

We know that the generator matrix of a systematic code has to be like this:

$G = \left[\begin{array}{c|c} I_{kxk} & G'_{kx(n-k)} \end{array} \right]$, where I_{kxk} is a kxk identity matrix.

We also know that the generator matrix and the parity check matrix of a code have to fulfill the following property:

$$G \cdot H^T = 0$$

If we can find a matrix G that matches these two equations we get a Gabidulin systematic generator matrix. This new code have the same properties as the non-systematic one, with a lower computational cost.

As we will see next, these systematic codes also allow us to construct product codes based on Gabidulin codes.

But the problem with Gabidulin codes is the constraint of $n \leq m$, which means that we can only send short messages. The solution we follow for this problem is to send a concatenation of encoded packets.

To try this concatenation of Gabidulin code words in a system that uses NC we propose the following scheme:

The network is constituted by 10 relays as shown in figure 4.

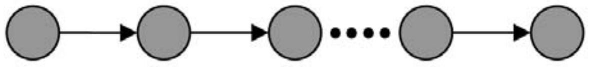
Figure 4. Diagram of a toy line network used in our simulations

Every information packet contains 300bits.

Every link is modeled by figure 5.

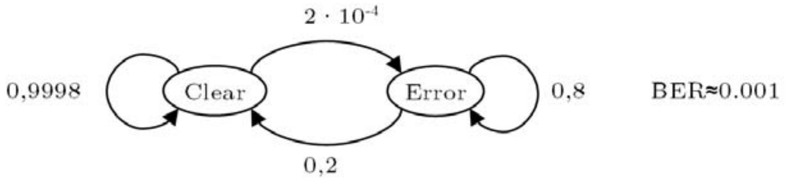

Figure 5. Gilbert-Elliot channel model used in our simulations. The probability of error in the clear state is zero, while the error probability of the error state sweeps from zero to one.

The error probability in the error state is one, while the error probability in the clear state is zero.

We simulate the system for different m, n, and k.

Good results are achieved when $m=11$ and $k=7$. To send 300bits we use 484bits. With these parameters 92% of packets are correct. If m is smaller we get a better correction, but the redundancy is increased.

2.4 Interleaving

Interleaving is frequently used in digital communication to improve the performance of forward error correcting codes. Many communication channels are not memoryless: errors typically occur in bursts rather than independently [4,5]. If the number of errors within a code word exceeds the error-correcting code's ability, it fails to recover the original code word. The main idea of interleaving is very simple: we take all the symbols of our message, we mix them all together and scramble them.

In order the receiver to be able to de-interleave, the scramble should not be completely random. Therefore, a pseudo-random interleaver is used. The pseudo-random interleaver produces a deterministic sequence for every seed, so, if the receiver knows this seed it can de-interleave correctly.

The drawback of interleaving is that it introduces some delay to the system. However, the gain in performance is big enough that outbalance this drawback.

As we will see in the following, this simple technique which barely adds complexity to the system, improves the performance of our systems significantly.

2.5 Product codes

Usually, some redundancy symbols are added to the source messages to protect them. Product codes use this procedure as well, but also create source messages full of redundancy symbols. This means that a vector of information symbols become a matrix of encoded symbols. The interesting thing about these codes is that if the horizontal decoding cannot decode properly, maybe the vertical decoding will do it. This is the reason why product codes are used together with iterative decoders.

We propose the configuration for product codes shown in figure 6.

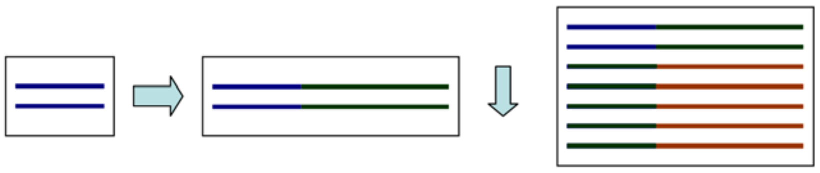

Figure 6. Schematic encoding procedure for product codes. The encoding in vertical and horizontal directions is depicted.

$$\begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 & x_2 & \tilde{x}_3 & \tilde{x}_4 \\ y_1 & y_2 & \tilde{y}_3 & \tilde{y}_4 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 & x_2 & \tilde{x}_3 & \tilde{x}_4 \\ y_1 & y_2 & \tilde{y}_3 & \tilde{y}_4 \\ \tilde{z}_1 & \tilde{z}_2 & \tilde{z}_3 & \tilde{z}_4 \\ \tilde{t}_1 & \tilde{t}_2 & \tilde{t}_3 & \tilde{t}_4 \end{bmatrix}$$

If we use our systematic Gabidulin codes to encode both directions we are able to use an iterative decoder that increases the performance of the system.

If the horizontal code is unable to correct an error then the vertical code may correct it and vice versa. This is why we need an iterative way to decode.

2.5.1 Iterative decoder

The iterative algorithm we propose works as following. First we decode vertically. If there are more errors than the correction capacity of the code, we leave the codeword as it is. If there are fewer errors than the correction capacity of the code, we decode and encode again. Then, we replace the old codeword with the new one.

Once we have decoded vertically we try to decode the horizontal direction following the same procedure. When we finish with the horizontal decoding we repeat the above procedure.

We repeat this iteratively until we cannot note any gain. Once it has converged we apply the normal decoding way to obtain the information word, thus we decode first vertically and then horizontally.

To compare the iterative procedure with the conventional one we try them in a system with these properties:

k = 2;   M = n = 4;   number of packets = 2;   BER = 0.1;

So, a 2x2 information matrix becomes a 4x4 matrix.

If we use the conventional decoding algorithm we get a 59% of correct packets, while if we use the iterative procedure we get a 72% for the same system parameters.

The problem with this decoding algorithm is that we need to know the error position to decode, which is not realistic. Then we need to find an iterative decoding procedure that doesn't need to know the error position.

We propose a very easy procedure:

We first decode the vertical direction and encode again. If more symbols than the correction capacity of the code have been changed we leave the codeword as it was, if not, we replace the old codeword by this new one. We then follow the same for the horizontal direction.

We iterate this until we cannot note any gain and then we apply the conventional decoding way.

If we try this new iterative decoding algorithm in the same system as before to compare both algorithms we find that this new one gets a 70% of correct packets, only 2% less than the old but without requiring any knowledge about the error positions.

2.5.2 Distributed product code

The encoding for these codes proposed in [2] is the same as for the product codes we presented in section 2.5 but using an RS encoder instead of MRD. These codes also use a different decoder which is similar to that of turbo-codes. This decoder is a Max-Log-Map decoder that generates soft information which is used to estimate the word most likely to have been sent. In fact, there are two Max-Log-Map decoders, one for the horizontal direction and another one for the vertical direction. This Max-Log-Map decoder compares the received word with all possible words that could have been transmitted. For each bit of the codeword the decoder finds the closest (according to hamming distance) codeword which has a zero in this bit position and the closest codeword which has a one in this bit position. With these two code-words and its respective hamming distances the decoder computes a soft output. The decoder does this for all bits in the received code words, and then, from all candidate code words, it computes the extrinsic information for the decoder in the other direction.

In the original decoder proposed in [2] the received word is not updated until the decoding algorithm is completed. Instead we propose to update the received code-word after every decoding step.

In other words, each decoder provides to the next one its extrinsic information and its estimated received word. This should make the algorithm faster and with even better performance.

These codes use Reed-Solomon as FEC code for both directions. What we propose is replacing it with Gabidulin codes. In order to be able to replace RS with MRD we have to make a change in the decoder. The decoder uses Hamming distance to compare the actual received codeword with the possible candidates, whereas in our decoder we will change Hamming distance with rank distance. With this simple change we get an iterative MAP decoder for MRD codes.

2.6 Rayleigh channels

Next, we briefly explain Rayleigh channels since they model wireless channel much better than the ones we have been using yet. Further information about Rayleigh channel can be found at [5, 10, 11].

The most common model for simulating channels is the so called Additive White Gaussian Noise (AWGN) channel. In this kind of channels a noise with a Gaussian probabilistic function is added to the signal. The reason why these channels are used is because they simulate quite well the effects of thermal noise in antennas and electric parts in systems. However for mobile channels another model that fits much more with the reality is the Rayleigh channel model. This model adds fading to signal, and eventually allows the modeling of different paths for the signal, which is what really happens in a mobile communication. Many times in a mobile communication, the signal is not received or acquired from the line of sight path but from reflections, and Rayleigh fading channel model models it.

Since we want to make our system reliable in a wireless environment thenceforward will try our systems over a Rayleigh channel.

2.7 Unequal Error Protection

This is a relatively new technique of forward error correction. It can be seen as a type of product codes but where different symbols are encoded with different redundancy, i.e. some symbols are more strongly protected than others. The reason why this technique has been applied recently is because in multimedia streaming some symbols are more important than others, so the most important symbols should be better protected than the less important ones. This allows us to send the same amount of information in less time and saving power. A good pointer for those readers who want further information about UEP coding and how to apply it is [12].

Chapter 3

Simulation results

3.1 Mesh networks

Until now we have tried our codes in a line network. But chain networks are not likely to be found in the real world. That's why we study our schemes in mesh networks. The network we consider for our trials is as shown in figure 7.

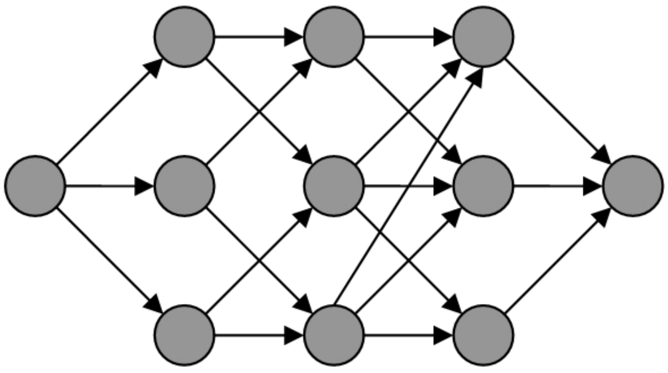

Figure 7. Toy mesh network used in our simulations. Every node is allowed to use network coding.

3.2 Comparison of Reed Solomon, MRD and product codes

We want to compare results of our proposed codes which use MRD as a FEC code or using RS. We also compare results between MRD, our proposed product codes and the uncoded case. We evaluate them in mesh networks where the nodes are allowed to perform network coding. For concatenated MRD codes we also examine the case when every node decodes and re-encodes to see whether increasing the computational complexity is beneficial.

The codes for all cases are defined over a Galois field GF(2^3), and the maximum codeword size is limited to $n = 3$, thus, the amount of information symbols per codeword is $k = 1$.

In the following we consider a model for every channel like the one shown in figure 8.

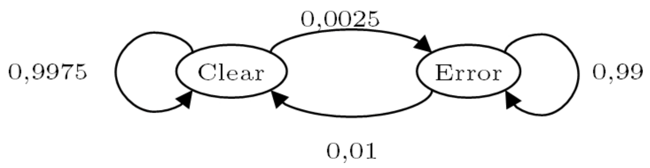

Figure 8. Gilbert-Elliot channel model for every link used in our simulations with the mesh network. The error probability of the clear state is zero and the error probability of the error state sweeps from zero to one.

The error probability in the clear state is zero. The error probability in the bad state sweeps from zero to one.

According to our results we realize that when the channel is too strong methods are inefficient, i.e., it doesn't matter if we use a

FEC code or not, as the error correcting capability of the codes is exceeded.

Although, when the channel allows the nodes to correct some packets, we notice that the best procedure to follow is to encode our symbols with the product code, followed by MRD coding. RS based scheme performs slightly worse that our MRD based and of course the worst results are obtained when we don't encode at all since there is no error correction ability.

What we want to do now is find out the gain coming from correction in every node of the network. As MRD codes are linear codes we can decode and re-encode in every node with no problems. In figure 3 we show the results for symbol error rate for both cases when we correct in every node and when we don't.

We observe that for small values of the bit error rate per channel we do get a considerable gain. For example when the $BER = 2\%$ the gain is about 25%. So we consider that correcting in every node is a good option.

A critical parameter of our systems is the size of the Galois field. On the one hand, if we choose a larger value for m *we* would be able to encode more symbols together, as m limits the codeword size. On the other hand, when the values for m are small the performance is better, since we are looking for performance we keep m as small as possible but bigger than 2 (which is the minimal m for Gabidulin codes), this means we try our systems for $m = 3$.

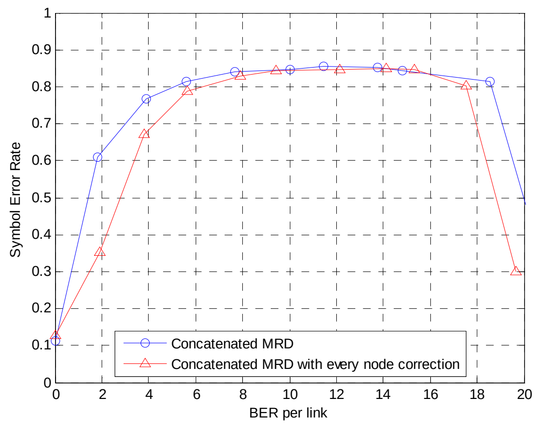

Figure 9. Performance in terms of end-to-end symbol error rate as a function of the bit error rate per channel of concatenated MRD codes when intermediate nodes are allowed to correct the received packets and when they are not.

3.3 Replacing erroneous symbols by zero

In case of product codes, if we replace erroneous symbols by a zero in every node we can reduce the error spread. In figure 10 we show the results for this scheme.

We observe that there is an improvement in the performance of the system. The problem with this scheme is that we assume that we know the position of the errors and we replace them with zero. But considering that we know the position of the errors is not realistic. Probably we could get a probabilistic decision of the positions using soft values but this would make our system more complex. So we won't adopt this method.

We also tried this method with an interleaver (we will explain this interleaver latter on). In this case we have good results, so interleaving is a procedure which we can consider in the following.

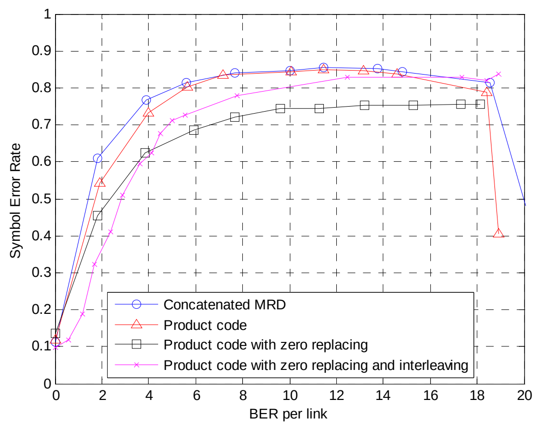

Figure 10. Symbol error rates at the receiver for different strategies. The figure compares the symbol error rate under four strategies: concatenated MRD codes, product codes, product codes replacing erroneous symbols by zeros, and product codes replacing erroneous symbol by zero combined with a random interleaver. It shows that product codes replacing erroneous symbols by zero combined with a random interleaver performs better than the others.

3.4 Product code with horizontal decoding in every node and using our iterative decoding algorithm at the destination

Now we examine our product codes but this time decoding the horizontal code in every node. This should minimize the error spreading since we have corrected some errors before they can be spread. At the receiver we use our iterative decoding algorithm.

To decode in every node we have to reverse the encoding procedure of our product codes in order to decode. Specifically, we have been encoding the horizontal direction first and the vertical direction second. Thus, from here on we first encode the vertical direction and then the horizontal one. With this, the rows of the encoded matrix, are valid MRD code-words, which allow us to decode and then encode again in every node.

In figure 11 we show the results for this last scheme. We show both bit error rate and symbol error rate. If we compare these results with the results we have for product codes we can see that there is a gain of about a 15% for reasonable values (between zero and 2%) for the BER. So correcting in every node should be applied in next schemes.

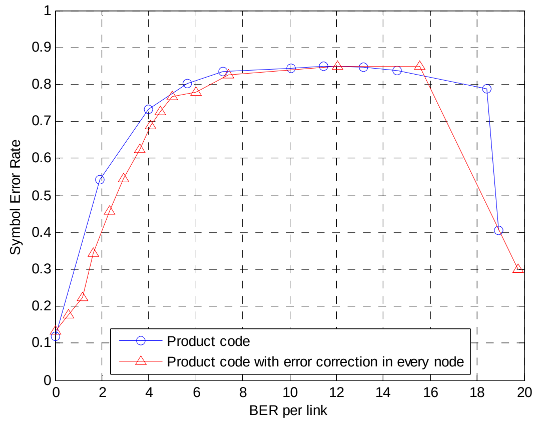

Figure 11. Symbol error rate at the receiver. Correction in every node of product codes improves the symbol error rate at the receiver. The figure shows the comparison between product codes with interative decoding algorithm at the receiver with product codes with iterative decoding algorithm at the receiver combined with a conventional decoding in every intermediate node.

3.5 Rectangular Interleaving

As we have seen, interleaving may improve the performance. That's why we follow some clever ways to use an interleaver in our systems.

The spreading of errors is always in the vertical direction of the matrix, so the vertical coding of our product codes doesn't help in the final decoding probability. To solve this issue we propose a special way of interleaving different sources.

Specifically, we first encode the horizontal direction of every source and then we interleave, using a fixed pattern. The main idea is that the symbols that correspond to the same information are sent through different sources. This pattern will spread a burst error in the vertical direction, hopefully improving the performance of the system.

Let us explain this with an example:

Imagine that we want to send the information symbols (x_1, y_1, z_1) *and we encode them with an MRD(3,3,1) code. Then we have the following encoded matrix,* $\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix}$*. It is sent over the network.*

For this example just imagine that there is only one node. This node performs network coding and interleaves the matrix as follows: $NC \Rightarrow \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \Rightarrow INTER \Rightarrow \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$*, what we should do is swap rows with columns. We consider symbols* $\widetilde{a}_2$ *and* $\widetilde{b}_2$ *are the* *erroneous. So the matrix the receiver receive is:* $\begin{bmatrix} a_1 & b_1 & c_1 \\ \widetilde{a}_2 & \widetilde{b}_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$. *The destination has to swap again rows for columns (de-interleave), and the resulting matrix results:* $\begin{bmatrix} a_1 & \widetilde{a}_2 & a_3 \\ b_1 & \widetilde{b}_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$. *Now we have to undo network coding, resulting that the resulting matrix just before the decoder is:* $\begin{bmatrix} x_1 & \widetilde{x}_2 & x_3 \\ y_1 & \widetilde{y}_2 & y_3 \\ z_1 & \widetilde{z}_2 & z_3 \end{bmatrix}$, *which is easily decodable for the MRD(3,3,1) code. If we hadn't interleaved, the resulting matrix prior decoding would have been:* $\begin{bmatrix} \widetilde{x}_1 & \widetilde{x}_2 & x_3 \\ \widetilde{y}_1 & \widetilde{y}_2 & y_3 \\ \widetilde{z}_1 & \widetilde{z}_2 & z_3 \end{bmatrix}$, *which is not decodable.*

The drawback of this scheme is that is applicable only to a chain network. We have been unable to find a way to apply it to a mesh network. The reason is because each node must have every packet and this only happens when the network is chain.

3.6 Random Interleaving

In the case of a mesh network and using product codes a random interleaving may help us improve performance, since the vertical coding should be able to decode more code-words.

First, we encode the information matrix vertically and horizontally, then we interleave each row, and then we send the new matrix through the network. To decode, we first deinterleave the received matrix and then we apply the iterative decoding procedure.

If we adopt this new procedure in our system we observe that we have a small improvement.

We find out that the performance improves when the words to be interleaved are larger. Then the best way to interleave the code words is first putting all the rows in a single vector, then interleaving and then reversing the conversion from matrix to vector. We make clear this with an example:

We want to send the symbol x_1 *and we encode it with a product code with an MRD(3,3,1) code as vertical and horizontal encoding code. So the resulting matrix to be sent is:* $\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix}$*. Then we convert this matrix to a vector,* letting us with this vector: $\begin{pmatrix} x_1 & x_2 & x_3 & y_1 & y_2 & y_3 & z_1 & z_2 & z_3 \end{pmatrix}$. We will now interleave this vector with a random interleaver, resulting in the following vector: $\begin{pmatrix} z_2 & x_3 & y_3 & y_1 & x_2 & z_3 & x_1 & y_2 & z_1 \end{pmatrix}$. Now we get the matrix to vector conversion undone. So the matrix that we send over the network is: $\begin{bmatrix} z_2 & x_3 & y_3 \\ y_1 & x_2 & z_3 \\ x_1 & y_2 & z_1 \end{bmatrix}$.

3.7 Double MRD interleaved

We have been encoding horizontally and vertically to obtain our product codes, which gives us more redundancy. This is not the only way to add redundancy.

To increase the redundancy of the codes and therefore to correct a larger number of errors without increasing the size of the Galois field, we propose the following scheme. Encode the words with a Gabidulin code, interleave and then encode again with the same Gabidulin code.

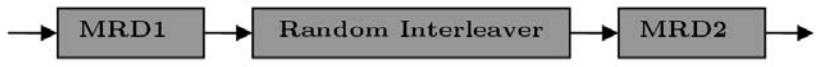

Figure 12. Block diagram used at the source node to encode. The horizontal direction is encoded twice allowing to have more redundancy symbols and, therefore, larger correction capacity.

The second Gabidulin code corrects the burst errors produced in the channel, and the first one corrects the errors that the other code is unable to correct.

We also try this system for decoding and re-encoding the second MRD code in every node. We show these results in figure 13.

The performance of the double MRD is the best performing end-to-end coding technique we got yet. The performance is close to the product codes replacing errors by zeros. Of course when we correct the MRD2 in every node the performance is even better.

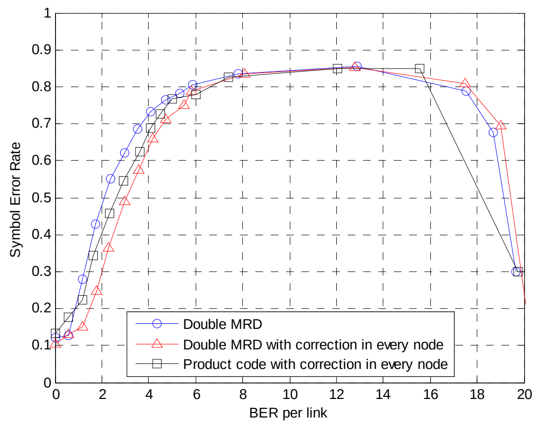

Figure 13. The figure plots the symbol error rate for three different strategies: using double MRD codes, double MRD codes allowing intermediate node to correct packets, and product codes when intermediate nodes can also correct packets and when the receiver uses an iterative decoding algorithm.

3.8 Combining RS and MRD

Previously we tried coding twice with MRD. The reason was that one of the codes could correct burst errors and the other correct the errors that the first one has been unable to correct. But this first code doesn't need to be good against burst errors, so we don't need to use an MRD code. Instead we can use an RS code, which allows us to increase the correction capacity of the code without having to increase the size of the Galois field. As an example, if $m = 3$ with an MRD(3,3,1) code we can correct 1 symbol, with a RS(3,7,1) we can correct 3 symbols. We add a lot of redundancy to the system, but we want to find out if it's worth it.

We also try this system correcting the MRD code in every node. The results for this are shown in figure 14.

We have introduced a lot of redundancy to our system, and as we expected the performance is increased. In fact this scheme is the best end-to-end system we have tried till this point. When we correct the MRD component in every node the percentage of correct symbols, for reasonable BER's, becomes acceptable.

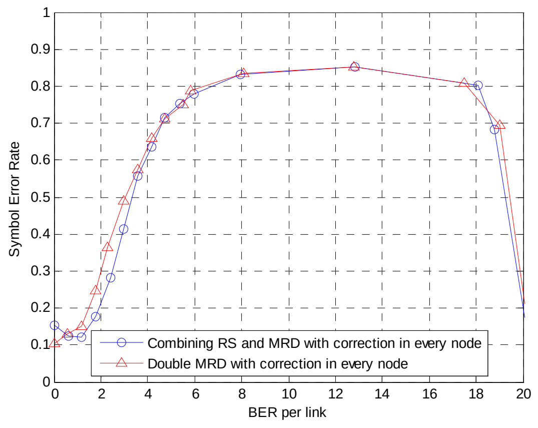

Figure 14. Symbol error rate comparison between two different techniques. In both techniques the intermediate nodes are allowed to correct packet. One scheme uses double MRD codes while the other uses as first encoder a RS encoder instead of an MRD encoder.

3.9 Distributed Product Code

The usage of turbo-like decoders for product codes used in [2] is interesting for us since we could apply it to our product codes. We adapt these decoders to our MRD product codes.

First we examine the performance of the decoder when we use it to decode a word after doing Gaussian elimination. The results for this scheme are as shown in figure 15.

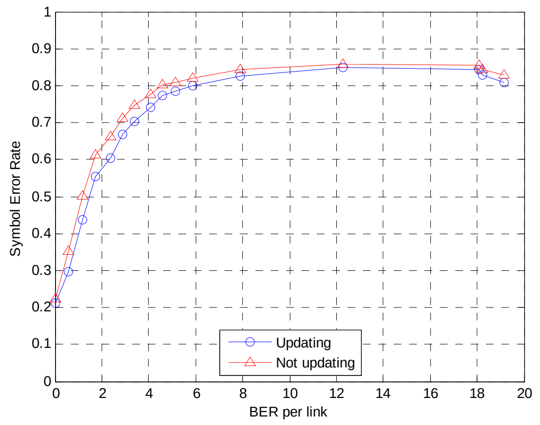

Figure 15. Decoder comparison of distributed product code when using network coding. Comparison of the decoder proposed in [2] with our proposal updating the received word before every iteration.

The blue line represents the performance of correct symbols after decoding with the iterative decoding but when the received word is updated in each step. The red line represents the performance for the iterative decoder without updating.

But, this decoder is not actually the decoder in [2] since in [2] they proposed decoding the received codeword before doing Gaussian elimination in order to reduce error propagation.

Above we have seen that the results are worse than those using Gaussian elimination first and then decoding. The drawback of this scheme is that the vertical code just before the decoder is the multiplication of an RS code with the NC matrix. The resulting code after this multiplication is not an RS code so we can't decode correctly the vertical component. To solve this issue we have to assure that the resulting vertical code have good hamming properties. To do that, we impose that the coefficient matrix in the last hop assures that the resulting code has a good hamming distance. When we test the performance of the system with this new constraint, it turns out that the performance for low BER's is lower than what we got previously. This is because now we are getting good matrices, in terms of hamming distances, for the RS codes, but these matrices are not good for NC. Therefore, we can conclude that the only way to get a good NC matrix and a good code just prior decoding is by increasing the size of the GF. The problem is that when we increase the size of the GF the performance drops. Then, we will first use Gaussian elimination and then decode.

We want to compare the results of this new iterative decoder with our previous iterative decoder. Our iterative decoder employed MRD codes, but with only a few changes we can adapt it to use RS. This will enable us to compare results. We have seen that the results for our iterative decoder with RS are better than the ones we get with these new iterative decoders.

To validate it we compare the decoders over one link without using NC. These should show us which decoder is best in error correcting terms. The comparison between these three decoders is shown in figure 16.

The red line shows the performance for symbol error rate for the iterative decoding without updating the received word. The blue line represents the performance for the iterative decoder when the received word is updated. And the black lines represent the performance for our iterative decoder.

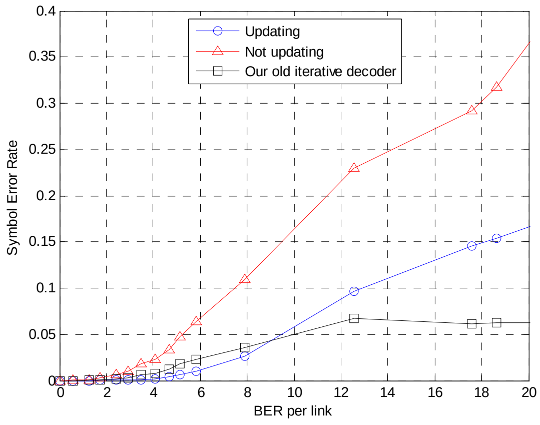

Figure 16. Evaluating different iterative decoders in one link. Comparison of our iterative decoder when received code words are updated in every iteration, the one proposed in [2] and our old iterative decoder where we evaluate the number of symbols switched before and after correction.

We can notice that our iterative decoder and the new iterative decoder have similar performance for reasonable values for the BER. The worst iterative decoder is the one where the received codeword remains unchanged until the final iteration.

As we can see we don't have really good results with these three decoders when we use NC since the best one has a performance of 43% of correct packets when BER=2%, so we conclude that, as MRD codes are better than RS codes, we may find a way to adapt these new decoders to work with MRD. For this reason we just have to change the metric, specifically, everywhere that hamming distance appears, we replace it with rank distance. With this simple change we get a Max-Log-Map decoder that works with MRD codes. If we now try this in our scheme we get the results shown in figure 17.

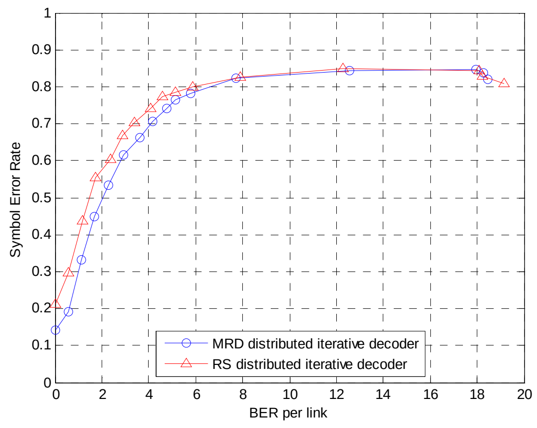

Figure 17. Symbol error rate comparison when using distributed product codes technique but replacing Reed-Solomon codes with Maximum Rank Distance codes.

Again the results are disappointing since for a BER=2% we have 50% of correct packets. To improve this we try increasing the GF where the NC is done (in order to have more linearly independent matrices) but keeping the size of the GF of the MRD codes as low as possible (to have better decoding performance). To explain this we use an example:

We have this vector $\begin{pmatrix} 5 & 1 & 3 & 2 & 4 & 7 \end{pmatrix}$ *where all the elements belong to* $GF(2^3)$*. We can express this vector with elements belonging to* $GF(2^6)$ *as* $\begin{pmatrix} 41 & 26 & 39 \end{pmatrix}$*.*

The results for this simulation are shown in figure 18.

The results when there are no errors in the network are better, since the resulting coefficients matrix results in independent code words, but the results are worse when there are some errors. This poor performance is because an error in a larger GF symbol produces more erroneous smaller GF symbols.

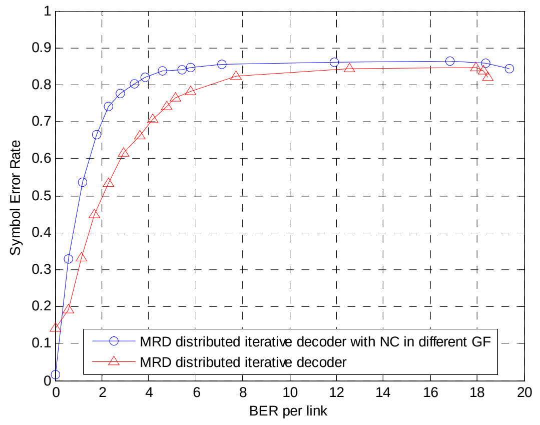

Figure 18. Performance comparison in terms of symbol error rate when using network coding combinations in a bigger Galois field than the one used in the MRD encoding.

An interleaving before doing NC could improve performance since the error spreading would be in different code-words.

In figure 19 are shown the results for this last proposal.

The performance is increased when there is low BER but still they are worse than the ones we got when the GF is the same for NC and MRD encoding.

This only lets us try different ways of sending the coded words. Our first thought is interleaving words.

With an interleaver we got better results, but still were not good enough.

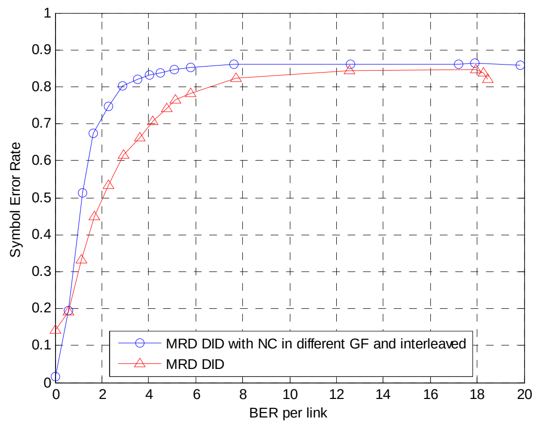

Figure 19. The figure plots the symbol error rate when using different Galois field for network coding operations and MRD coding combined with a random interleaver. It is compared with a scheme where MRD coding and network coding operations are in the same Galois field.

We now reconsider our iterative decoder. The way it works is: first, it decodes the received codeword and encodes it again; and then it compares the resulting corrected word with the received word. If more symbols than the correction capability of the code are changed it keeps the original received codeword. If less symbols are changed it takes the new corrected codeword. This means that if the decoder can help to correct the received word it may improve the performance. If it can't help, it lets the received codeword as it was. This is really interesting for us, because this means we can try to decode first with our iterative decoder and then decode with the new iterative decoder. So, somehow our iterative decoder will assist the new one to decode. We also use an interleaver. A diagram for the system could be as shown in figure 20.

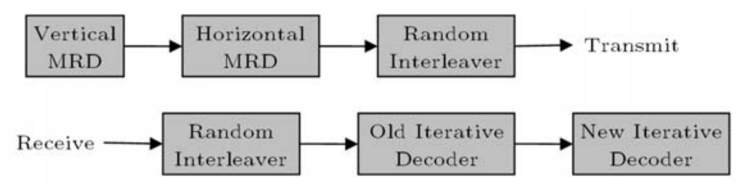

Figure 20. Encoder and decoder block diagram of product codes using an interleaver and using both iterative decoders at the destination.

The results for this combined iterative decoder are shown in figure 21.

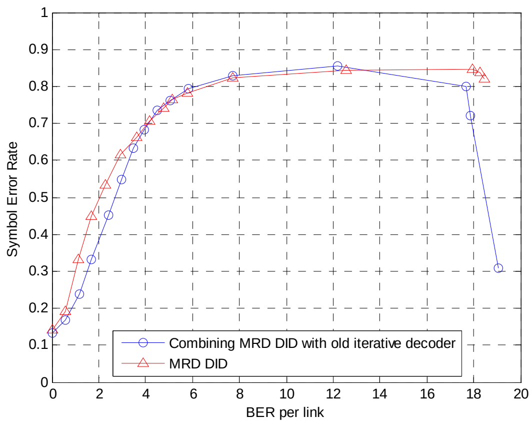

Figure 21. Performance comparison when using both iterative decoders at the receiver and when only the distributed iterative decoder is used. Combining both iterative decoders improves the performance of the system.

Again we improve results, but still we are not happy with them. The proposed system in [2] uses decoders in every hop of the network, so we think we could do it as well. We have used decoder in each hop before but not yet with this new combined iterative decoder. The decoders in every node will only decode the horizontal direction. To keep the horizontal direction being a valid MRD codeword we can't use the interleaver we have been using. But we think that if somehow we could use an interleaver and yet having valid code words in the horizontal direction we could have even more gain as it would allow us to decode and re-encode in every node. The problem is how to make the receiver have valid product code words at the input of the combined iterative decoder. The block diagram shown in figure 22 explains our proposal.

The issue is that the interleaver at the sender and the deinterleaver at the receiver are not quite the same. We show it with an example:

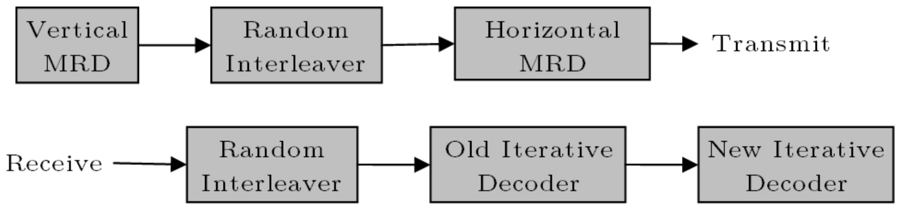

Figure 22. Encoder and decoder block diagram for product codes which can be corrected in every node, using a random interleaver, and using both iterative decoders at the destination

Imagine we want to send a symbol, for example a 4. We encode the symbol vertically and we obtain the MRD word $(4 \;\; 3 \;\; 5)'$*, then we interleave this resulting codeword randomly. We get something like this* $(5 \;\; 3 \;\; 4)'$*, for example. Then we encode the horizontal direction and we obtain the resulting product code word, that in our example is this one:* $\begin{pmatrix} 5 & 1 & 3 \\ 3 & 6 & 1 \\ 4 & 3 & 5 \end{pmatrix}$*. This is what we send over the network.*

At the receiver, after doing Gaussian elimination we have, if no errors occur, this very same word. But this word is not decodable with our decoder, as the words in the horizontal direction are valid MRD code words whereas the words in the vertical direction are not. So we have to deinterleave first, and this deinterleaving is not at a symbol level, but instead it will be at a codeword level, which *in our example means exchanging rows. So after deinterleaving we will have this codeword:* $\begin{pmatrix} 4 & 3 & 5 \\ 3 & 6 & 1 \\ 5 & 1 & 3 \end{pmatrix}$ *and this word is a valid product code word so now we can decode it with our iterative decoders.*

As now we do have valid code words for the horizontal direction in every node we can decode them. We show the results for this scheme with decoder in each hop and without it in figure 23.

Finally we got good results, at least for low BER. This, then, is our proposal. Next we try it with a Rayleigh fading channel which models better a wireless channel better.

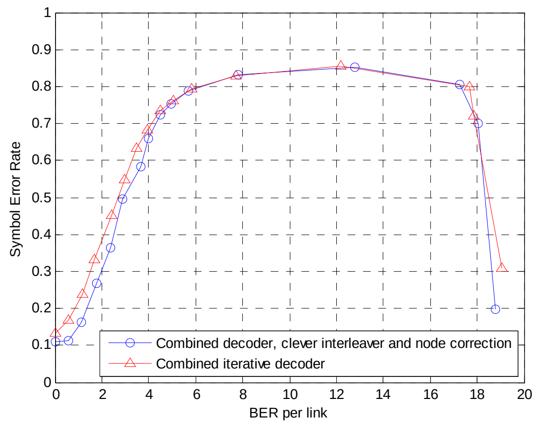

Figure 23. The figures shows the symbol error rate combining both iterative decoders at the receiver when using an interleaver and correction is allowed in every intermediate node of the network and when intermediate nodes can only forward the received packets.

3.10 Using Rayleigh fading channel

We will now try our last scheme in a Rayleigh channel to see if our system could work well in a wireless environment.

The parameters we choose for the channel are:

- Modulating frequency: 1Mhz
- Maximum Doppler spread: 100Hz

We use a BPSK modulation for the symbols and we try the performance for different SNR (signal to noise ratio).

Figure 24 shows the results for our system over this kind of channel.

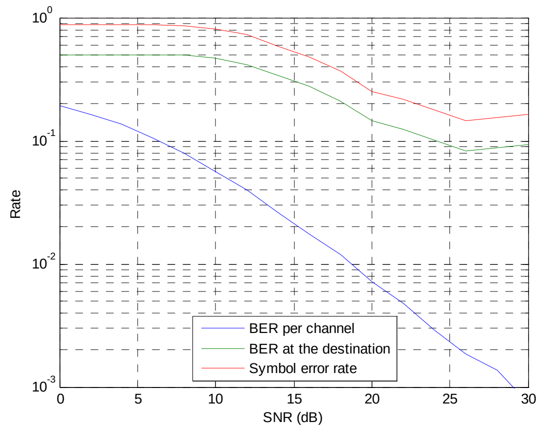

Figure 24. Distributed product code with an interleaver, decoding in every node and using a combined iterative decoder over a Rayleigh channel. The figure plots three different lines: one for the BER per channel, one for the BER at the receiver, and another one for the symbol error rate at the receiver.

We observe that when the SNR is higher than 20dB our system performs well.

3.11 Unequal Error Protection

To try UEP we implemented the following scheme: We used Gabidulin codes over the Galois Field GF(2^5). We used two different codes, one with correction capacity 2 and another one with one. To explain how we use them to encode a codeword we show it with an example:

We want to encode the vector of information symbols $\begin{pmatrix} x & y & z & t & v \end{pmatrix}$*. Now we reorganize this vector into a 5x5 matrix like this:* $\begin{bmatrix} x & y & z & * & * \\ t & * & * & * & * \\ v & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}$*. The next step is encoding the vertical direction (the first column with MRD(5,5,3) and the second and third with MRD(5,5,1)). And the last step is encoding the horizontal direction with MRD(5,5,3). So the resulting encoded matrix is a 5x5 matrix.*

These are the code words we send through the network. But with UEP we can't use correction in every node because combining two different codes means that the resulting code words no longer belong to a linear code. At the receiver we use, as always, both iterative decoders.

The problem with this scheme is that the amount of possible code words that the decoder has to check is huge, taking a lot of time to simulate. However, the fact that we cannot use decoding in every node is a big drawback of using UEP.

In order to run a simulation involving UEP we tried encoding only in one direction and using conventional decoding. The codes involved are the same as above. With this new configuration we get the results shown in figure 25.

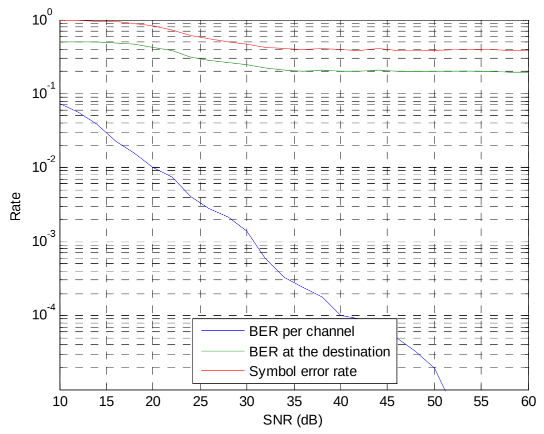

Figure 25. The figure shows the results for our proposed UEP scheme. Three lines are plotted for different rates: one for BER per channel, one for BER at the receiver, and one for symbol error rate at the receiver.

As we can see, the results are not as good compared to the ones we got with other configurations. We also need to mention that the complexity of the decoding of this last case is much simpler than before, so a direct comparison wouldn't be accurate.

Conclusions and future work

The aim of this work was to find a way to do reliable multimedia streaming using network coding. In concluding this work we think that our solution improves the one proposed in [2] by keeping the operating complexity level low. Furthermore, we have proposed a way of building a systematic Gabidulin code which has allowed us to construct product codes based on MRD codes instead of RS. Besides, we have also built an iterative decoder which helps in the decoding of product codes when used with a turbo-like iterative decoder. We have shown that some traditional methods, such as interleaving, when used with network coding techniques, improve the performance of the whole system.

Despite our work has found a good way to do multimedia streaming, we are aware that there is still a lot of work to be done. For those who want to follow this work here are some guidelines for improvement.

- The MAP iterative decoder used is just a modification of the one with RS. We think we could get better performance with a new iterative decoder based on MRD.
- Our system should be tried in a multipath fading channel.
- It should also be tried with a real multimedia stream.
- Try to change the searching algorithm for the iterative decoder in order to reduce the time it takes to run.
- Also a way of getting close to 100% of clean symbols when there are no errors in the channel should be done. Maybe a mathematical method to obtain linearly independent matrices in small Galois fields can be found. If not, maybe there is a subset of matrices that perform well in network coding.

References

[1] *Symbol-level Network Coding forWireless Mesh Networks*
Sachin Katti, Dina Katabi, Hari Balakrishnan, and Muriel Medard

[2] *A Distributed Product Coding Approach For Robust Network Coding*
Jingyao Zhang, K. B. Letaief, and Pingyi Fan

[3] *Network Coding and Media Streaming*
Nikolaos Thomos and Pascal Frossard

[4] *Information Theory and Network Coding* by Raymond W. Yeung, The Chinese University of Hong Kong
Springer, August 2008

[5] *Digital Communications* by John G. Proakis and Massoud Salehi
McGraw Hill, January 2008

[6] *The new construction of rank codes*
Alexander Kshevetskiy and Ernst Gabidulin

[7] *A rank-metric approach to error control in random network coding*
Danilo Silva, Frank R. Kschischang, and Ralf Kötter

[8] *On metrics for error control in network coding*
Danilo Silva and Frank R. Kschischang

[9] *Error and erasure decoding of rank-codes with a modified Berlekamp-Massey algorithm*
Gerd Richter and Simon Plass

[10] *Rayleigh fading channels in mobile digital communications systems Part I: Characterization*
Bernard Sklar

[11] *Rayleigh fading channels in mobile digital communications systems Part II: Mitigation*
Bernard Sklar

[12] *Optimized Transmission of JPEG2000 Streams over Wireless Channels*
Nikolaos Thomos, Nikolaos V. Boulgouris, and Michael G. Strintzis

[13] *Coding for errors and erasures in random network coding*
Frank R. Kschischang and Ralf Kötter

[14] *Adversarial error correction for network coding: models and metrics*
Danilo Silva and Frank R. Kschischang

[15] *Using rank-metric codes for error correction in random network coding*
Danilo Silva and Frank R. Kschischang

[16] *Rank metric decoder architectures for noncoherent errors control in random network coding*
Ning Chen, Maximilien Gadouleau, and Zhiyuan Yan

[17] *Fast encoding and decoding of Gabidulin codes*
Danilo Silva and Frank R. Kschischang

[18] *XORs in the air: practical wireless network coding*
Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft

[19] *Reed Solomon Codes*
Ralf Köetter

[20] *Theory of Codes with Maximum Rank Distance*, Problems of Information Transmission, v. 21, No. 1, pp. 3-14, 1985.
E. M. Gabidulin