Best of Both Worlds: Prioritizing Network Coding Without Increased Space Complexity

Roman Naumann, Stefan Dietzel, and Björn Scheuermann
Humboldt-Universität zu Berlin, Berlin, Germany
Email: roman.naumann@hu-berlin.de, stefan.dietzel@hu-berlin.de, scheuermann@informatik.hu-berlin.de

***Abstract*—Random linear network coding simplifies routing decisions, improves throughput, and increases tolerance against packet loss. A substantial limitation, however, is delay: decoding requires as many independent linear combinations as data blocks. Hierarchical network coding purportedly solves this delay problem. It introduces layers to decode prioritized data blocks early, which may benefit video streaming applications or applications for sensor information collection. While hierarchical network coding reduces decoding delays, it introduces significant space complexity and additional decoding time. We propose a decoding algorithm that manages all prioritization layers in a joint decoder matrix. Analytical evaluation and performance measurements show that we maintain prioritization benefits without increased space complexity and improve decoding performance. With memory requirements independent of the number of layers, our algorithm facilitates more fine-grained prioritization layers to further the benefits of hierarchical network coding.**

***Index Terms*—network coding; random linear network coding; hierarchical network coding; prioritization; decoding**

I. Introduction

Many applications that require robust network transmissions can benefit from network coding [1]. Network-coded packets differ from normal packets in that their content is a combination of two or more original data packets. With random linear network coding (RLNC), the encoder crafts and transmits a random linear combination, while the decoder collects linear combinations until it can solve a system of linear equations. Solving the linear system yields the original data. Network coding differs from other coding techniques (such as raptor codes [2]) in that intermediate nodes are able to re-encode packets they receive instead of just forwarding them.

While numerous areas, such as, content distribution [3], wireless sensor networks [4], and video streaming [5], have been shown to benefit from RLNC, the technique is not yet applicable when a low transmission delay is key, as the decoder cannot solve an underdetermined linear system. In other words, for n encoded packets, the decoder requires n independent linear combinations in order to decode all n packets. This decoding delay is a price associated with RLNC, whereas in other systems the receiver can immediately process individual packets as they are received.

To mitigate the decoding delay, Nguyen *et al.* [6] propose hierarchical network coding (HNC) in the context of video streaming networks. HNC is an extension of RLNC: the authors propose to divide a video stream into three layers; the first layer provides basic video quality, and the other layers provide quality increments. The receiver can decode the first layer as soon as it received $n/3$ linearly independent packets from this layer. At the same time, packets that encode layer one are also valid linear combinations for decoding the other layers. In addition to video streaming, HNC can be applied to all applications that can prioritize information and benefit from early reception of more highly prioritized information. For example, information that is sampled by remote sensors can be transformed using Discrete Cosine Transform (DCT) to obtain similar enhancement layers [7].

However, no specialized decoding algorithm has yet been proposed for efficient and incremental decoding of HNC's layers. HNC layers can be decoded without a specialized algorithm by employing regular RLNC decoding algorithms and running them in parallel, once for each layer. But this approach introduces significant space complexity and additional computational complexity.

We contribute the Joint Layer Decoder (JOYCE), a novel decoding algorithm for HNC. Our analytical evaluation and performance measurements show that space usage and decoding time are strictly better than HNC and, under practical conditions, as good as RLNC. Most importantly, our approach improves memory requirements for hierarchical network coding from $\mathcal{O}(|R| \cdot n^2)$ to $\mathcal{O}(n^2)$, where $|R|$ is the number of prioritization layers. Our approach's memory requirements are independent of the number of prioritization layers and, hence, identical to RLNC, thereby facilitating highly layered HNC systems, which can be used to further minimize decoding delay. Furthermore, the low space complexity opens new applications for HNC in resource-constrained environments such as wireless sensor networks.

As our contribution focuses on efficient decoding of layered network coding systems, we do not change how the source encodes linear combinations or how intermediate nodes re-encode information, nor do we introduce additional network overhead. In fact, we do not require any modification at all to existing on-wire formats.

Next, we review related work in Section II. Section III explains our system model and introduces an example use case. Section IV gives an overview of all components necessary for HNC's efficient decoding; the details of our proposed decoding algorithm are discussed in Section V. We assess our proposed algorithm analytically and measure performance of an example implementation in Section VI. Section VII concludes the paper.

This article was originally published in: R. Naumann et al., "Best of Both Worlds: Prioritizing Network Coding Without Increased Space Complexity" in *Proc. 2016 IEEE 41st Conf. Local Computer Networks (LCN)*, 2016 © IEEE. doi: 10.1109/LCN.2016.123

II. Related Work

We survey related work in two relevant areas. First, our decoding algorithm's domain are network coding systems. We, therefore, give an overview of existing work on network coding and related approaches to delay minimization. Second, we survey existing work on solving linear equation systems to select a suitable solving mechanism as basis for our decoder.

A. Network coding

Ahlswede *et al.* [8] proposed network coding as a means to improve throughput in communication networks. In contrast to previous mechanisms, network coding allows intermediary nodes to re-encode information as it is forwarded. The most common form of network coding is RLNC [9], where linear combinations are built with random coding coefficients. These linear combinations form a system of linear equations that is solved by the sink to retrieve the original data.

Wireless ad hoc networks benefit from applying RLNC, because it improves bandwidth use while reducing routing complexity [4]. As such, it is an alternative to previous approaches that require to calculate distance metrics between sources and sink and require to maintain routing tables [10]. Besides ad hoc networks, the idea of network coding has been applied to peer-to-peer content distribution networks [11]–[13] and multimedia streaming [14].

A drawback of network coding in these scenarios is that it is not generally possible to decode a subset of data blocks before enough linear combinations are received to decode all blocks. In ad hoc networks, storing large partially solved linear systems may exceed available node capacity. In peer-to-peer file sharing, and even more so in multimedia applications, the delay caused by waiting for enough linear combinations may be prohibitive for delivering sufficient service quality.

The delay issue has been acknowledged early on in network coding research. Chou *et al.* [15] describes an information dissemination scheme for ad hoc networks where more important information is coded with higher redundancy and can be decoded earlier by receiving nodes. Later, Nguyen *et al.* [6] coined the term hierarchical network coding (HNC) to include different priorities in network coding. Chau *et al.* [16] extend HNC to use hierarchical coding layers. In their protocol, the source divides a video stream into a base layer and two quality enhancement layers. The first layer is represented using linear combinations where only the first third of all n coefficients are used and the remaining encoding vector is filled with zeros. Using the proposed encoding format, the receiver can decode the first layer as soon as it received $n/3$ linearly independent packets from this layer, because the linear combinations that pertain to a layer form a linear subspace. Thus, the layering concept helps to decode parts of the whole information with lower delay. The authors, however, do not discuss how decoding such layered information can be implemented efficiently. If all layers are decoded using separate systems of linear equations, the amount of memory required for decoding increases linearly in the number of layers, which limits the system's applicability.

B. Solving linear equation systems

Solving linear systems is necessary to retrieve original information in RLNC. Nodes transmit packets that are composed of an encoding vector, i. e., coefficients, and an information vector, i. e., coded information. Gaussian elimination [17] writes entries from the encoding and information vectors as rows into an extended coefficient matrix, the decoder matrix [18]. When the decoder matrix has full rank, the matrix is brought into upper triangle (i. e., row echelon) form and then solved using backward substitution. Gaussian elimination requires $\mathcal{O}(n^3)$ floating point operations to bring a matrix into upper triangle form and $\mathcal{O}(n^2)$ operations for the backward substitution. Although RLNC reuses coefficients for blocks of K symbols [3], both steps have to be performed K times, once for each symbol in a block. Rather than performing backward substitution once the matrix has full rank, it can be done incrementally when new packets are received [5], [13].

LU decomposition is "a 'high-level' algebraic description of Gaussian elimination" [19, p. 111]. To solve the linear system $Ax = b$, LU decomposition works in two stages: first, the coefficient matrix A is factored into a lower and an upper triangular matrix L and U, respectively, so that $A = LU$. Second, the utility system $Ly = b$ is solved by backward substitution and $Ux = y$ is solved by forward substitution. The first step takes $\mathcal{O}(n^3)$ floating point operations, the second and third step only require $\mathcal{O}(n^2)$ operations. Solving coefficients independent of information vectors is particularly beneficial for network coding, because encoding vectors are usually reused for a number of symbols from the original information. In LU decomposition, the (computationally more expensive) first step only has to be performed *once* instead of K times, where K is the block size; only the less expensive substitutions need to be performed K times. Therefore, we base our decoding algorithm on LU decomposition, which we extend to support decoding of prioritized layers in a joint decoding matrix.

III. System Model

We propose an advanced decoding algorithm for HNC that may benefit a range of different applications. Namely, use cases where several layers of information are to be transmitted with different priorities can apply our decoding algorithm. Example applications are

- video streaming services with different layers that can be combined to improve video quality;
- wireless sensor networks that may want to prioritize alarm conditions but also transmit additional fine-grained process information in other layers; and
- peer-to-peer game distribution, where prioritized blocks may enable playing the first level earlier.

We already discussed an example for video streaming in Section II-A. Here, we introduce an industrial wireless sensor network as a second example use case, which illustrates benefits of prioritization and hierarchical network coding, and which we use to describe our algorithm in Section III-A. Yet, our algorithms are not specific to these use cases and can be applied to other settings. Based on the example use case, we derive an abstract information model in Section III-B that is used as input to the network coding algorithms.

A. Example Use Case

Consider an industrial wireless sensor network where multiple sensors attached to manufacturing machines transmit data to a single sink. The sink uses sensor information to control and optimize the production process. In our scenario, machines operate in so-called production cycles. That is, machines produce individual parts and the production of each part constitutes a cycle during which sensor information is obtained and transmitted. The need for prioritization arises from two application scenarios for the collected information. Preliminary analysis of sensor information can indicate misconfiguration or a defective machine: an emergency signal needs to be issued to avoid production of scrap or damage to the machine. This high-priority function can operate with a rough approximation of the sensor values. In addition, complete sensor information may be used for parameter optimization. In this case, information should be exact but delays are more tolerable. Depending on the factory layout, information may be transmitted – using network coding – over multiple communication hops. The resulting topology is a multi-hop, multicast network.

Summarizing, we identify three main characteristics in our example scenario, which equally apply in similar settings:

1) the need for improved robustness against transmission faults, better throughput, or reduced protocol complexity, which can be provided by network coding;
2) availability of layered information with prioritization and delay requirements that cannot be fulfilled when entire network coding generations are decoded; and
3) limitations on the available storage, memory, and computational capacity of the devices in the network.

Chachulski *et al.* [4] demonstrate that RLNC improves throughput and reduces routing complexity in our example and similar scenarios. However, systems that employ RLNC introduce additional delay at three points:

- *Encoding* requires time beyond just sending a packet.
- *Decoding* itself takes additional processing time.
- *Partially decoding* information is not generally possible before at least n linear combinations were received.

HNC has been proposed as an extension to RLNC that addresses the *partial decoding* delay, but neglects the increase in *decoding* delay that it introduces.

B. Information Model

Next we generalize and formalize the information items that are transmitted. RLNC requires a partition of input data into generations and a partition of each generation into data blocks. No coding takes place between different generations' data blocks. Without loss of generality, we consider one generation and a single information source for the remainder of this paper. To support multiple subsequent generations, our decoding mechanism simply runs repeatedly; multiple sources can be modeled by running multiple decoders in parallel.

We assume that the source aims to transmit a number of individual symbols. We organize symbols into n blocks of K symbols each for later encoding; the k-th symbol of the i-th block is M^i_k with $1 \leq k \leq K$ and $1 \leq i \leq n$. Symbols are represented as elements of a finite field $\mathbb{F}_q$. In practice, we often have $M^i_k \in \mathbb{F}_{2^8}$, that is, each symbol represents one byte of information. A vector of symbols $M^i = (M^i_1, M^i_2, \ldots, M^i_K)$ forms the i-th information block. The set of all blocks $M = \{M^1, M^2, \ldots, M^n\}$ comprises all information.

As discussed in Section III-A, we assume that some blocks should be transmitted with higher priority (and lower delay) than others. In previous work [7], we have demonstrated how DCT can be applied to sensor information to divide it into different layers with incremental quality. Similarly, for most forms of video or audio streaming applications, a suitable prioritization already exists in the form of a base layer and enhancement layers.

Without loss of generality, we assume that the blocks in M are ordered by decreasing priority. That is, M^1 contains the most important information; M^n contains the least important information. We identify layers by a set of integers $R = \{r_1, \ldots, r_{|R|}\}$. As layers are assumed to incrementally improve information quality, each r_i identifies the number of blocks required to represent the *first i layers* (rather than the i-th layer). Thus, we require $\forall i < j : r_i < r_j$ and $r_{|R|} = n$.

Both the number of layers $|R|$ and how they are distributed across the blocks are highly application-specific choices. For some video streaming applications, few layers – e. g. a base layer and a small number of layers providing video quality increments – may provide sufficient flexibility. Other applications, such as the protocol discussed in [7], benefit from a large number of layers to provide many subsequent increments of information quality.

IV. Efficient Hierarchical Network Coding

A network coding system consists of three algorithmic components: the encoder, the re-encoder, and the decoder [18]. We now discuss encoding and re-encoding in Section IV-A, reiterating necessary basics of network coding where useful. Our main contribution is an efficient decoder component for hierarchical network coding, which we introduce conceptually in Section IV-B. In Section V, we discuss our decoding algorithms in more detail. As an example throughout this section, we will consider a configuration with $n = 4$ and two layers: $r_1 = 2$ and $r_2 = 4$.

A. Encoding and Re-Encoding

To transmit information to the sink, the source creates a sequence of network-coded packets, the j-th packet having the form (c^j, X^j). The components are:

1) an encoding vector $c^j = (c^j_1, c^j_2, \ldots, c^j_n)$, that is, a sequence of randomly chosen coefficients over $\mathbb{F}_q$ and
2) an information vector $X^j = (X^j_1, X^j_2, \ldots, X^j_K)$, which is a sequence of coded symbols.

Each coded symbol is a linear combination of data symbols:

$$X_k^j = \sum_{i=1}^{n} c_i^j M_k^i \quad \forall 1 \le k \le K. \tag{1}$$

For our mechanism, we extend encoding in the same way as HNC does [6]. In essence, HNC prioritizes layers by creating coded packets that only encode blocks contained in a subset of all layers in addition to packets that are coded using all blocks. The first layer in our example, $r_1 = 2$, only encodes information from the first two blocks; $r_2 = 4$ encodes the first four blocks.

Encoding of packets for each layer is achieved by setting all remaining coefficients to zero: in a packet for the p-th layer, we set $\forall i > r_p : c_i = 0$. Otherwise, encoding is identical to RLNC. Omitting the zero coefficients, each coded symbol becomes:

$$\forall 1 \le k \le K : X_k^j = \sum_{i=1}^{r_p} c_i^j M_k^i. \tag{2}$$

Because only r_p linearly independent coded packets are required to decode the data blocks of layer p, the layered encoding effectively leads to earlier decoding of prioritized packets at the sink. How early the sink can decode the received information depends on the strategy that the source uses to send coded packets for different layers. For instance, the source may alternate between layers for each transmission, it may first send a number of high priority layer packets, or it may randomly select layers. We will discuss decoding efficiency in more detail in Section IV-B.

Re-encoding at intermediate nodes is identical to RLNC, but packets of different layers should not be mixed.

B. Decoder Overview

To accommodate for space efficient decoding of many hierarchical layers with low storage space overhead, we propose a new decoding algorithm that is based on LU decomposition and a joint decoding matrix. Here, we provide an overview of our proposal's key algorithmic ideas. Section V will provide details of our implementation and a decoding example.

Assume that the receiver collected m coded packets $(c^1, X^1), (c^2, X^2), \dots, (c^m, X^m)$. Each packet contributes a linear equation to K systems of linear equations:

$$X_k^j = \sum_{i=1}^{n} c_i^j M_k^i \quad \forall 1 \le j \le m, 1 \le k \le K. \tag{3}$$

Each of these systems has n unknowns, $M_k^1, \dots, M_k^n$, and m knowns $X_k^1, \dots, X_k^m$. A core requirement to decoding layered network coding is that the receiver must maintain its decoding state in such a way that, as soon as r_p linearly independent packets that encode the p-th layer have been received, the blocks contained in those layers can be reconstructed.

To solve all K equation systems simultaneously, we use LU decomposition, as discussed in Section II-B. As received packets may encode different layers of the original data, we need to amend the orifinal LU decomposition algorithm to

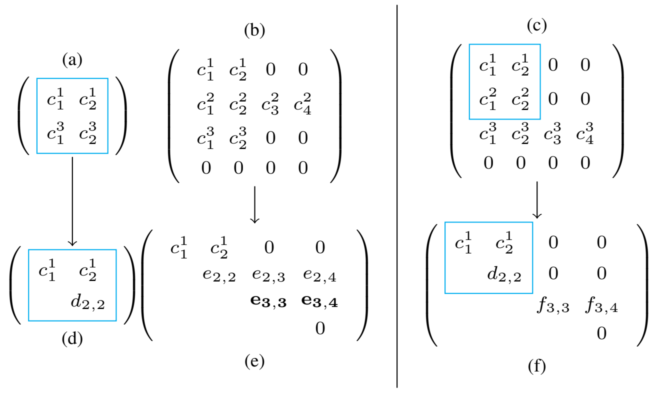

Fig. 1. Decoder state: existing vs. proposed algorithm.

support joint decoding of different layers. Figure 1 shows an abstract comparison between existing solutions and our algorithm. In the example, the sink has already received three coded packets, two of which belong to layer r_1 and one belongs to layer r_2. In the example's matrices, we use known identifiers for entries as much as possible, e. g. c_2^1. When the content of an entry is unknown, we introduce a new identifier named after its matrix, e. g., element $(3, 3)$ in matrix (f) becomes $f_{3,3}$.

The received information suffices to successfully decode layer r_1. Without modifications, however, the sink would need to maintain separate decoding matrices for each layer, in Figure 1 denoted by (a) and (b), which need to be solved in parallel, resulting in (d) and (e) after factorization was applied. Matrix (d) solves layer r_1 while (e) is the (yet incomplete) solution towards decoding r_2.

Our contribution is to allow for early decoding of individual layers without the need for additional decoding matrices. Instead, we regard the matrices for each individual layer as sub-matrices of a joint-decoding matrix (c). Without modification, applying LU decomposition to (b) destroys the different layers' encoding vectors' trailing zeros. This effect is shown in (e). Intuitively, matrix (e) does not contain enough rows with zeroes in columns 3 and 4 to decode layer r_1, because trailing zeroes ($e_{3,3}$, $e_{3,4}$) were overwritten during Gaussian elimination. Therefore, the equation in row three of matrix (e) no longer forms a linear subspace with the first row, prohibiting partial decoding.

To enable joint decoding, we extend LU decomposition to maintain the order of layers during decoding, as shown in matrix (f), where entry $(2, 2)$ equals $d_{2,2}$ after factorization. Matrix (f) can be used to decode layer r_1, while at the same time it is the currently optimal, incomplete solution towards decoding r_2. Note that the example in Figure 1 is a simplification since no decomposition has been applied to matrices (a), (b), and (c) previously. Our decoder, in contrast, works incrementally and supports partially decoded input matrices for the reordering step, which we describe in the next section.

V. Decoding Algorithms

Implementing the joint decoder matrix requires two main algorithms. First, we extend LU decomposition to allow for *incremental decoding* steps whenever new packets are received. Second, we implement a method we term *counter elimination*, which inverts individual steps of the LU elimination algorithm to allow for order preservation – and thereby early decoding – of individual layers. Third, we describe an optimization that introduces two utility structures ϕ and δ to refine the decoder's comparison function. This optimization, according to our experiments, has such a drastic effect on performance that we consider it an integral part of the decoding algorithm.

A. Initialization

A core aspect of LU decomposition is that the coefficient matrix is first solved independently before the solution is used to recover the original data packets. We extend the algorithm to solve the coefficient matrix without affecting the remaining steps to recover the original data.

LU decomposition uses three matrices: a pivotization matrix P, a lower triangular matrix L, and an upper triangular matrix U. Initially, U is a zero-filled $n \times n$ matrix, and L and P are $n \times n$ identity matrices.

When the first packet (c^1, X^1) is received, c^1 is used to fill the first row in U; L and P are not modified, and X^1 is stored for later steps of the LU decomposition algorithm. No information can be decoded yet, except when $r_1 = 1$, in which case the received information trivially represents the first information block M^1.

B. Incremental Decoding and Reordering

Whenever an additional coded packet is received, we update the decoding matrices and incrementally decode the so-far-received information to ensure that information in prioritized layers can be decoded as early as possible.

The incremental decoding algorithm uses the decoding technique described by Fragouli *et al.* [18]. We extend the existing incremental approach by (1) modifying it for use with LU decomposition and (2) by maintaining a joint decoder matrix for prioritization layers $R = \{ r_1, r_2, \ldots, r_{|R|} \}$. To implement joint decoding, it is necessary that rows in U, i.e., encoding vectors, are sorted by their associated prioritization layer. Otherwise, the trailing zeroes in their encoding vector would be overwritten during the elimination process, preventing the decoder from decoding prioritized data blocks early. For now, we focus on the implementation of incremental decoding using LU decomposition; we discuss the details of the reordering process in Section V-C.

Algorithm 1 shows the high level decoder algorithm. The loop in lines 3 to 4 writes the new encoding vector in U's first free (i.e., all-zeros) row. Depending on the received packet's prioritization layer, U's rows may not be ordered by priority after the insertion. Therefore, we call the procedure ReorderByLayer (line 5) to reorder the rows according to their encoding vector's prioritization layer. Finally, Gaussian elimination (line 6) is used to eliminate elements in U and write the factors used to L. As usual, elimination may perform row swaps as part of its pivotization, so the order of the rows may change again at this step.

Algorithm 1 Update decoder for new encoding vector v.

```
1: procedure Update($v$)
2:     $u \leftarrow$ current matrix rank
3:     for all $i \leftarrow 1, 2, \ldots, n$ do
4:         $U_{u+1,i} \leftarrow v_i$
5:     ReorderByLayer($P, L, U$)
6:     GaussianElimination($P, L, U$)
```

Algorithm 2 Adjacent-row-swap in-place comparison sort.

```
1: procedure ReorderByLayer($v$)
2:     repeat
3:         $s \leftarrow$ false
4:         for all $i \leftarrow n-1, n-2, \ldots, 1$ do
5:             if $\neg\,(i \blacktriangleleft i+1)$ then
6:                 Swap($i, i+1$)
7:                 $s \leftarrow$ true
8:     until $\neg s$
```

The procedure ReorderByLayer in Algorithm 2 implements a comparison-based in-place sorting algorithm. Essentially, the comparison operator $\blacktriangleleft$ is used to (re-)order U so that encoding vectors are ordered by the priority of their layers, starting with the most important vectors at the top. Swaps perform the actual exchange of encoding vectors in U whenever out-of-order rows are found. Both comparisons and swaps need to take into account the particularities of LU decomposition to avoid side effects; most importantly, they must update L and P to reflect changes in U, and they must ensure that no zeros are swapped to U's main diagonal.

To this end, we exploit that our decoder works incrementally; that is, whenever a packet is received, the matrix is in an almost-sorted state. When a new coding vector is added at the bottom of U, it is likely out of order. Hence, we start sorting with the last non-zero row in U. We use the encoding vectors' layers to decide whether two adjacent rows should be swapped. The layer information can be derived from an encoding vector by counting its number of trailing zeros. The helper data structure γ maps row indices to their encoding vector's layer. Formally, two rows i and j where $i < j$ need to be swapped unless $\gamma_i \leq \gamma_j$:

$$i \blacktriangleleft j \Leftrightarrow \gamma_i \leq \gamma_j \tag{4}$$

Algorithm 3 implements the actual swapping of two adjacent matrix rows. The cell swaps in matrices U and L are performed in lines 5 to 8. But before doing so, the swap operation needs to "undo" certain elimination operations so that swapping rows i and j does not invalidate the decoder state. Most importantly, note that the row swap operation changes the entries in the matrix' main diagonal for rows i and j. Cells on the main diagonal may have been used for elimination, but $U(i,i)$'s new position may itself be subject to

Algorithm 3 Swap operation for rows i and $j = i + 1$.

```
1: procedure SWAP($i, j$)
2:     $u \leftarrow$ current matrix rank
3:     for all $k \leftarrow j+1, j+2, \ldots, u$ do
4:         COUNTERELIMINATE(k,i)
5:     SWAPROWS($P$,$i$,$j$)
6:     SWAPROWS($U$,$i$,$j$)
7:     for all $k \leftarrow 1, 2, \ldots, i$ do
8:         $L_{i,k} \leftrightarrow L_{j,k}$
```

Algorithm 4 Counter elimination.

```
1: procedure COUNTERELIMINATE($i, j$)
2:     for all $l \leftarrow 1, 2, \ldots, N$ do
3:         $U(i,l) \leftarrow U(i,l) + L(i,j) \cdot U(j,l)$
4:     $L(i,j) \leftarrow 0$
```

elimination after the swap. Therefore, lines 3 to 4 processes all cells in column i below the swapped rows. For each of those cells, we apply what we term *counter elimination* to undo previous elimination operations and prepare the row swap.

Moving through U in decreasing row order (Algorithm 2 line 4), the comparison – and possibly swap – is repeated with adjacent pairs of i, j where $j = i + 1$ until no out-of-order rows are found in the mutable part of U.

C. Counter Elimination

To implement counter elimination and "undo" certain elimination operations, we need to re-calculate previous values in U and alter L accordingly. Effectively, counter elimination is the partial reversal of a previous iteration of Gaussian elimination. The technique eliminates a factor in L and modifies U accordingly – whereas Gaussian elimination eliminates in U and modifies L. Counter elimination as shown in Algorithm 4 applies to a cell in U and L that is selected by row and column index i, j, respectively. In lines 2 to 3, the original values of U are restored using the information contained in L. Afterwards, line 4 discards the corresponding elimination factor in L.

Once counter elimination has been performed on all necessary cells, and all row swaps have been performed, the high-level algorithm continues to apply Gaussian elimination. Barring pivotization, U is still ordered by increasing layer priority afterwards and allows to decode information from all layers for which sufficient encoding vectors have been received. When receiving the next packet; the decoding algorithms are executed in the same way until all information layers have been recovered.

D. Optimized Comparisons

In Equation (4), we defined the comparison function $\blacktriangleleft$ solely in terms of γ without delving into optimization details. Our experiments show that this definition, while functionally correct, results in bad performance. Here, we explain necessary changes to the comparison condition to achieve high performance.

Gaussian elimination requires non-zero elements on U's main diagonal; when the elimination algorithm encounters a zero on the main diagonal, it performs row swaps as part of its pivotization. The simple swap condition $\gamma_i \leq \gamma_j$ reverts these pivotization swaps even if no rows that substantiate the need for pivotization changed. In other words, Algorithm 1 line 5 induces row swaps that are immediately undone in line 6, rendering the swap operation useless in the first place.

Our optimized comparison avoids superfluous row swaps based on two utility structures ϕ and δ. The first utility structure, ϕ, is a marker that tracks which parts of U may be modified momentarily and which parts are immutable. Whenever a new encoding vector is received, ϕ is reset to point to the new vector's position in U; all rows with a lower row index are considered immutable. At least one row in a swap operation must be mutable, otherwise the swap is skipped. Whenever two rows $i, i + 1$ are swapped, we update $\phi \leftarrow \min(\phi, i)$. By adhering to these rules, rows are swapped only when the reordering is not necessarily undone by subsequent Gaussian elimination.

Skipping row swaps based on ϕ alone, however, introduces corner cases where our decoder cannot decode prioritized layers as early as possible. Specifically, pivotization may occur in the Gaussian elimination phase so that two encoding vectors i and j are not ordered by their respective prioritization layer. A linear combination with layer less than $\max(\gamma_i, \gamma_j)$ normally suffices to resolve such pivotization. With ϕ-immutability in place, though, these out-of-order rows cannot be resolved by new linear combinations that belong to a prioritization layer greater or equal to $\min(\gamma_i, \gamma_j)$. We therefore define a second utility structure δ to ensure that pivotization can be resolved in all cases; δ maps a row index i to the maximum layer found in all rows with an index smaller or equal to i:

$$\delta_i = \begin{cases} \gamma_i, & \text{if } i = 1 \\ \max(\gamma_i, \delta_{i-1}), & \text{otherwise.} \end{cases} \tag{5}$$

Based on ϕ and δ, we define the optimized comparison operation as follows:

$$i \blacktriangleleft j \Leftrightarrow j < \phi \vee \delta_i \leq \gamma_j. \tag{6}$$

Using δ instead of γ in the right inequality in Equation (6) enables swapping the new encoding vector below a point where pivotization occurred. Hence, an attempt is made to resolve pivotization during the next Gaussian elimination phase.

E. Decoding by Example

Now that we have explained JOYCE's decoding details, we complement this with an example that focuses on the interaction of the utility structures in particular: Figure 2 shows the main decoder matrix U in different decoding stages (a)–(g); P and L are not shown in the example. In addition to U, we show the utility structures δ, γ, and ϕ left of each matrix. U is a 6×6 decoder matrix, but we only show the non-zero rows for conciseness. The dotted line indicates that rows above this row index are set immutable by ϕ. We do not show the cells'

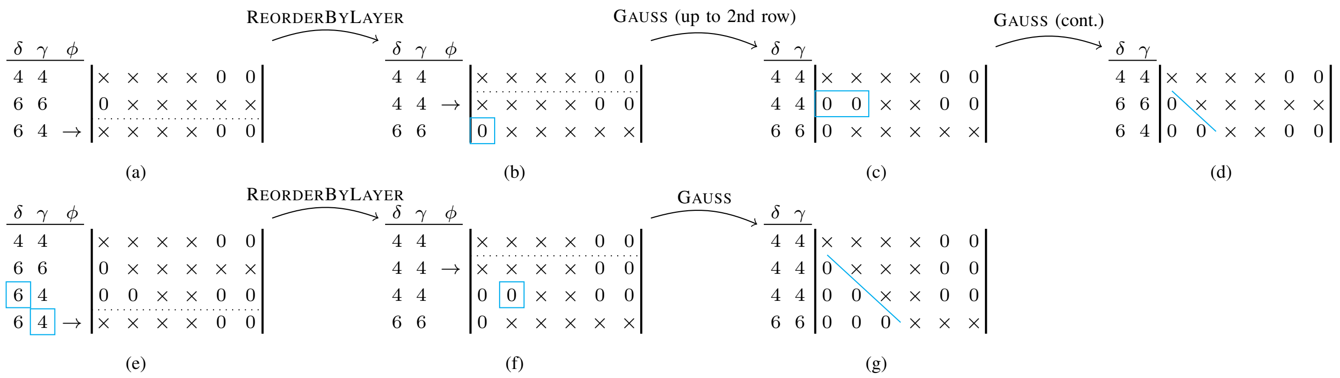

Fig. 2. Decoding by example.

exact content, but “0” for a finite field’s neutral element or, otherwise, the placeholder symbol “$\times$”. The example coding system considers two layers, $r_1 = 4$ and $r_2 = 6$.

Initially, in state (a), the decoder previously received and processed two linear combinations with layers r_1 and r_2. The third, newly received encoding vector is written to row 3 and belongs to layer r_1. Comparing δ_2 ($= \gamma_2$) to γ_3, the call to ReorderByLayer swaps rows 2 and 3. State (b) shows the result, in which order by prioritization layer is restored. In state (b), the already-eliminated cell ($U_{3,1}$) is still eliminated, i. e., the decoding work performed in previous increments was preserved by the swap. Next, Algorithm 1 line 6 invokes GaussianElimination. After eliminating $U_{2,1}$, $U_{2,2}$ also becomes zero in our particular example, as can be seen in state (c). Since $U_{2,2}$ lies on the main diagonal and must not be zero, Gaussian elimination invokes pivotization and reverses the previous row swap, which is shown in state (d), where the decoder matrix U is in upper triangular form.

In state (e), another linear combination of layer $r_1 = 4$ is received. Comparing δ_3 to γ_4 indicates that rows $(3, 4)$ need swapping in an attempt to resolve the pivotization. After swapping rows $(3, 4)$, the procedure ReorderByLayer swaps rows $(2, 3)$ and finally rows $(3, 4)$, resulting in decoder state (f). State (f) reveals that cell $U_{3,2}$, despite being subject to counter elimination, retains its zero value. After another application of GaussianElimination, matrix U in decoder state (g) is upper triangular. In addition, the matrix is ordered by prioritization layer. If another linear combination of layer $r_1 = 4$ is received, the first layer can be decoded utilizing the first three rows’ already-decoded state.

VI. Evaluation

To evaluate our algorithm, we assess it analytically and using performance measurements of an example implementation. We compare its decoding delay, computational complexity, and memory requirements with both RLNC and a hierarchical decoder without joint decoding matrix. As discussed in Section III, we assume a single sender and a single generation of information. Moreover, we assume fine-grained layering (up to one layer per symbol) may be used by the sender; that is, $|R| \in \Theta(n)$. Finally, we assume that the sender uses a packet scheduling strategy that sends coded packets by order of priority. Packet order may, however, be imperfect due to incomplete knowledge about the receiver’s state, which is influenced by a lossy transmission channel. We model this assumption as a random deviation from the correct order that is bounded by a constant C. More specifically, the sender transmits a number of packets that encode the layer r_{ideal}, that is, the minimum layer that helps decode information, with a random offset uniformly distributed in the range $[-C, +C]$.

A. Analytical Evaluation

We first discuss the *decoding delay*. Assume that the original information is divided into $|R| \in \Theta(n)$ priority layers, and we are interested in the delay until the first layer’s r_1 packets can be decoded. With RLNC, the decoder must receive at least n linearly independent packets before it can decode the first layer. Using HNC, only r_1 linearly independent packets that encode r_1 are required. When packets are sent in order, our algorithm’s decoding delay is in $\Omega(n/|R|)$; if we assume fine-grained layering (i. e., $|R| \in \Theta(n)$), partial decoding delay is in $\Omega(1)$. Using our decoder does not change this property.

As described in Section II, the dominant factor for *space complexity* during decoding RLNC is the $n \times n$ coefficient matrix; space complexity is $\mathcal{O}(n^2)$ [19, p. 116]. Decoding HNC layers separately requires $|R|$ parallel decoders with a total space complexity in $\mathcal{O}(|R| \cdot n^2)$. When using fine-grained layers, we have space complexity $\mathcal{O}(n \cdot n^2) = \mathcal{O}(n^3)$. Our approach uses a joint decoder matrix for all layers. The only additional data data structures are two mappings of size n from row index to prioritization layer, which are used for efficient reordering in the decoder matrix. As these data structures are asymptotically insignificant compared to the space of the decoder matrix, our decoding algorithm has space complexity $\mathcal{O}(n^2)$, as good as RLNC and much better than HNC.

It is harder to find an upper bound on *computational complexity* than space complexity. LU decomposition requires $\mathcal{O}(n^3)$ operations to transform a matrix into L and U. HNC incrementally performs n insertions of linear combinations into the matrices. Each insertion requires inserting into up to $|R| \leq n$ decoding matrices, each taking $\mathcal{O}(n^2)$ floating point operations. HNC’s computational complexity, therefore, is in $\mathcal{O}(|R| \cdot n^3)$, which is up to $\mathcal{O}(n^4)$ for fine-grained layers.

To find an upper bound on computational complexity for JOYCE, we discuss the steps that are performed in addition to normal RLNC decoding. We take a top-down approach

TABLE I
COMPLEXITY OVERVIEW

Algorithm	Delay	Space complexity	Computational complexity
RLNC	$\Omega(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
HNC	$\Omega(1)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$
JOYCE	$\Omega(1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$

and start with Algorithm 1: JOYCE extends the incremental elimination process by a reordering step. This step takes place during an increment that is called $\mathcal{O}(n)$ times in total. Considering that we need $\mathcal{O}(n)$ increments, the total cost of JOYCE is $\mathcal{O}(n^3 + n \cdot C_R)$ where C_R is the cost associated with each call to REORDERBYLAYER.

REORDERBYLAYER is essentially a sorting algorithm that implements comparisons and swaps. In our implementation, comparisons are much cheaper than swaps. Comparisons require a constant number of operations, whereas row swaps consist of two for loops: one that performs the actual value swaps and one that performs counter eliminations. In the worst case, our reorder algorithm performs a quadratic number of swaps. However, we assume that information is received in order with a maximum deviation of C layers. As sorting is only performed on the out-of-order part of the matrix U, we have at most C^2 swaps, which is in $\mathcal{O}(1)$ with respect to n. The swap operation itself contains one loop that is iterated n times to perform the actual value swap. In addition, counter elimination is performed up to C times. Counter elimination itself iterates over all columns and therefore requires $\mathcal{O}(n)$ operations. This gives us $C_R \in \mathcal{O}(n)$ and a total computational complexity for JOYCE in $\mathcal{O}(n^3)$.

Table I summarizes the results of the analytical evaluation. Using our decoder combines the best properties of RLNC and HNC: early decoding with low space complexity and low computational complexity. Of course, asymptotic complexity provides only a rough picture of the actual run time of algorithms, especially since practical values for n are small in actual network coding implementations. Hence, we complement our analytic evaluation with performance measurements.

B. Performance Evaluation

We compare the performance of JOYCE with HNC decoding that uses separate decoding matrices per layer (hereafter called "HNC") and normal RLNC.

All algorithms are implemented in C++ and compiled by Clang v 3.7.1 using optimization level 3. We performed tests on a Linux system with kernel version 4.4.7, an Intel Core i7 processor, and 8 gigabytes of main memory.

To ensure that the process image and all libraries are in the file-system cache, we performed two preliminary runs before each measurement. The measurements themselves were repeated five times for each set of parameters. We refrain from implementing optimizations for the individual algorithms' intricacies to ensure comparability. All data points in plots show the arithmetic means and 95% confidence intervals. Error bars might not always be visible in the figures when the error is very small. Test data and network coding coefficients were generated using C++'s `random_device` random number generator. Memory measurements show the *peak resident size,* performance measurements only measure the time it takes to bring the coefficient matrix to upper triangle form, i. e., the first step of LU decomposition, since the second step is identical for all three decoders.

1) Memory Usage: For memory usage, we fixed n and varied the number of prioritization layers, since those have a direct effect on the number of matrices for HNC. Memory usage for HNC and JOYCE is shown in Figure 3. The plot confirms Section VI-A's analytic results: HNC's memory usage grows linearly with the number of prioritization layers, whereas our approach's memory usage is independent of $|R|$.[1] For applications that benefit from fine-grained layers (i. e., $|R| = n = 384$), we reduce memory usage, on average, by a factor of 19.

2) Decoding Time: As described in Section VI-A, the decoding time depends on the order in which the different prioritization layers' linear combinations are received by the decoder. In Figure 4, we use the same sender model as in Section VI-A and assume that the sender has only limited knowledge about the decoder's state, for example, due to packet loss. Since the layered approaches HNC and JOYCE transmit more linear combinations than non-layered RLNC, we measure cumulative decoding time. That is, we measure the time that *all* insertions take until full rank is archived.

HNC's performance is near-identical for different values for C; we exemplarily plot $C = 16$. The parameter $C = 16$ means that linear combinations encode a number of data blocks that differs from the ideal layer by up to 16, which we consider realistic for most applications. The parameter $C = 64$ yields an offset range of 128, which is already one third of n and represents very coarse information about the decoder state.

Our results show that, for both $C = 16$ and $C = 64$, our algorithm performs significantly better than HNC. In systems using fine-grained layering, $|R| = n = 384$, JOYCE performs on average 166 times better than HNC when $C = 64$ and 185 times better when $C = 16$. Compared to RLNC, our approach runs 4% faster when $C = 16$ and 23% slower when $C = 64$. The first result is surprising, because RLNC does not use any prioritization layers. We would, therefore, expect RLNC to always perform better than JOYCE. We attribute our approach's performance gain to the simplified elimination process, as many of the cells in U are zero when prioritization layers are used.

Finally, we evaluate performance in a worst-case scenario where the sender randomly chooses layers to use for creating linear combinations. That is, the sender's uncertainty is not bounded by C in this setting. We can see in Figure 5 that JOYCE's performance degrades significantly in comparison to Figure 4. However, even in this worst-case scenario, JOYCE performs consistently better than the HNC decoder. We plot RLNC for comparison, although no prioritization is supported.

Summarizing, our measurements confirm the complexities

[1]HNC has fewer data points, as its implementation requires n to be a multiple of $|R|$.

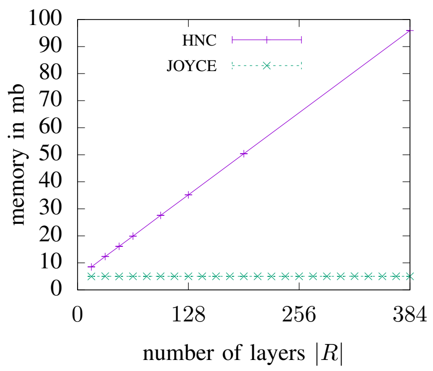

Fig. 3. Peak memory usage.

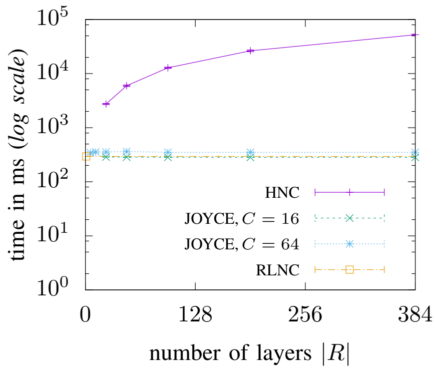

Fig. 4. Cumulative decoding time.

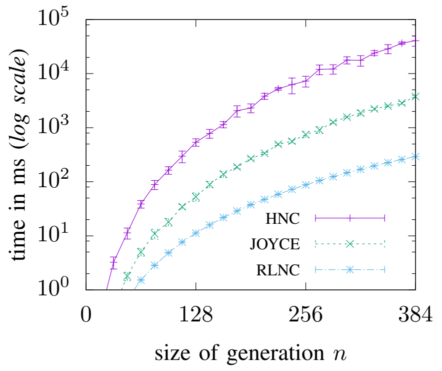

Fig. 5. Worst case cum. dec. time.

derived in Section VI-A. Memory consumption of our approach is constant, whereas HNC has linearly increasing memory requirements with respect to the number of prioritization layers. Performance of our approach is two orders of magnitude better than HNC and equal to RLNC when we have limited knowledge about the sender state. This means that in most practical scenarios, our decoding algorithm can be used to add prioritization support to RLNC systems without incurring cost, neither computationally nor memory wise. Even in the worst-case scenario, JOYCE performs significantly better than HNC, although in this case, prioritization comes at the cost of additional decoding complexity.

VII. Conclusion

We implemented an efficient decoding mechanism for coded packets that represent different prioritization layers in hierarchical network coding. We use LU decomposition to decouple decoding the coefficient matrix from solving the linear system, which improves decoding performance. In contrast to existing work, we specifically support a joint decoding matrix for all prioritization layers by implementing row swap operations that preserve priority order and allow for early decoding of prioritized information. Our analytical and simulative evaluation show that memory usage and performance of our decoder are both strictly better than with a non-specialized HNC decoder.

If we can assume that the sender has limited knowledge about the receiver's state, our approach's performance is on par with RLNC, which does not provide prioritization. In this case, our decoder allows prioritization without incurring additional decoding cost when compared to non-prioritized RLNC. Regardless of the order in which packets are sent, memory usage is independent of the number of prioritization layers used. Our system therefore facilitates systems that use more fine-grained prioritization layers to further the benefits of HNC and opens new application areas where the memory usage of HNC has been prohibitive.

Acknowledgments

We thank Holger Doebler for helpful discussion on reordering in partially factorized matrices. This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 636892 in the context of the PREVIEW project.

References

[1] R. Yeung, "Network coding: A historical perspective," Mar. 2011.

[2] A. Shokrollahi, "Raptor codes," 2006.

[3] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, Mar. 2005.

[4] S. Chachulski, M. Jennings, S. Katti, *et al.*, "MORE: A network coding approach to opportunistic routing," 2006.

[5] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, May 2007.

[6] K. Nguyen, T. Nguyen, and S. c Cheung, "Peer-to-peer streaming with hierarchical network coding," in *2007 IEEE International Conference on Multimedia and Expo*, Jul. 2007.

[7] R. Naumann, S. Dietzel, and B. Scheuermann, "INFLATE: Incremental wireless transmission for sensor information in industrial environments," in *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, Dec. 2015.

[8] R. Ahlswede, N. Cai, S. Y. R. Li, *et al.*, "Network information flow," Jul. 2000.

[9] S.-Y. Li, Q. Sun, and Z. Shao, "Linear network coding: Theory and algorithms," Mar. 2011.

[10] S. Biswas and R. Morris, "ExOR: Opportunistic multi-hop routing for wireless networks," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM 2005, ACM, 2005.

[11] S. Acedanski, S. Deb, M. Médard, *et al.*, "How good is random linear coding based distributed networked storage," in *Workshop on Network Coding, Theory and Applications*, 2005.

[12] D. Wang, Q. Zhang, and J. Liu, "Partial network coding: Theory and application for continuous sensor data collection," in *14th IEEE International Workshop on Quality of Service*, Jun. 2006.

[13] X. Chu and Y. Jiang, "Random linear network coding for peer-to-peer applications," Jul. 2010.

[14] E. Magli, M. Wang, P. Frossard, *et al.*, "Network coding meets multimedia: A review," 2013.

[15] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," 2003.

[16] P. Chau, S. Kim, Y. Lee, *et al.*, "Hierarchical random linear network coding for multicast scalable video streaming," in *Asia-Pacific Signal and Information Processing Association, 2014 Annual Summit and Conference (APSIPA)*, IEEE, 2014.

[17] S. C. Althoen and R. McLaughlin, "Gauss-jordan reduction: A brief history," 1987.

[18] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: An instant primer," 2006.

[19] G. Golub and C. Van Loan, *Matrix computations*, ser. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013.