Hierarchical Layer Selection with Low Overhead in Prioritized Network Coding

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

***Abstract*—Network coding simplifies routing decisions, improves throughput, and increases tolerance against packet loss. A fundamental limitation, however, is delay: decoding requires as many independent linear combinations as data blocks. Prioritized network coding reduces this delay problem by introducing a hierarchy of prioritization layers. What remains is the problem of choosing a layer to approach two often-contradicting goals: reduce delay until prioritized layers can be decoded and keep the total number of transmissions low. In this paper, we propose an algorithm for this problem that – based on limited feedback – primarily minimizes per-layer delay but identifies opportunities to reduce the required transmissions when per-layer delay is unaffected. Our evaluation shows that our algorithm improves per-layer delay compared to hierarchical network coding and is close to the theoretical optimum number of total transmissions.**

I. Introduction

Network coding (NC) is a widely studied approach to communication systems [1]. Originally introduced by Ahlswede *et al.* [2] as a technique to improve the throughput in networks, NC has been proven to benefit many fields since, e. g., peer-to-peer applications [3]–[5], network streaming [6], and combinations thereof [7]. NC also improves robustness, which means that the system better copes with packet loss. Distributed algorithms can be simplified, and link capacities can be saturated more effectively [1]. To implement these benefits, NC breaks with traditional routing paradigms. Namely, nodes combine two or more incoming packets and send these newly built combinations instead of just forwarding the original packets. In this paper, we discuss an improvement for the most broadly studied class of network coding, linear network coding, where original packets are combined into *linear* combinations [8].

One restriction inherent to NC is that it introduces additional delay. With high probability, a receiver cannot retrieve any of the original content as long as the number of received linear combinations is lower than the number of messages that were combined [9], [10]. Prioritized network coding (PNC) [11] builds on linear network coding and reduces decoding delay by introducing a hierarchy of priority layers on the original messages. Linear combinations are computed per priority layer rather than using the full message set. Consequently, a receiver is able to decode a prioritized subset of messages with fewer linear combinations. This encoding technique is also known as expanding window random linear coding [12]. The order in which different layers' linear combinations are sent has a direct impact on individual layers' achievable decoding performance and principal decodability at any given point in time [13], [14]. In order to achieve a reduced delay and at the same time avoid significant additional overhead, combinations have to be built and sent in a sequence that reflects the layers' individual priorities.

ISBN 978-3-903176-08-9 © 2018 IFIP

In summary, PNC can reduce per-message delay, but it only does so under the right circumstances: all senders need to carefully choose the correct layers for use in their next linear combination. Otherwise, linear combinations either have an increased chance of being linearly dependent, which results in increased message overhead, or they introduce additional per-packet decoding delay, which results in less effective prioritization. This selection problem of PNC has not yet been studied in detail for scenarios with limited knowledge about the receivers' decoder state. Existing approaches select layers uniformly at random (e. g., [11], known as hierarchical network coding (HNC)), or they use a weighted random choice, giving higher weight to prioritized layers [12]. Both strategies cause overhead, since they result in an increased probability of non-innovative content being sent.

In this paper, we take a more structured approach to the selection problem and analyze – based on limited feedback messages –, which selection of layers (a) reduces the total number of transmissions and (b) improves per-message decoding delay. We derive performance indicators that allow a node to classify layer choices based on these two criteria and propose an algorithm, eNhanced Prioritized nEtwork Coding (iNsPECt), which implements a deterministic strategy for choosing the layer that is used to generate the next linear combination. Based on these performance indicators, we instantiate a wireless multi-hop protocol that is based on single-hop feedback messages, which concisely summarize each node's decoder matrix state. Our protocol thereby implements prioritization with much less overhead than existing approaches while retaining the low decoding delay.

The remainder of this paper is structured as follows. Section II discusses related work, and Section III introduces our system model. In Section IV, we approach the layer selection problem with two performance indicators that guide layer choices. We further introduce an algorithm that yields optimal results under an analytical model, which we instantiate as a practical network protocol in Section V. Our simulative evaluation compares the protocols iNsPECt, HNC, and NC in Section VI before Section VII concludes the work.

II. Related Work

NC was introduced by Ahlswede *et al.* [2] to improve throughput in communication networks. In their work, the network model is a directed graph with one node as the source and multiple nodes as receivers. Ahlswede *et al.* demonstrate that the optimal throughput, which is given by the "minimum cut" between the source node and any receiver in a network graph, can be achieved when the nodes send linear combinations of the original messages. Later, Ho *et al.* [9] showed that randomly chosen linear coefficients $c_1, c_2, \ldots$ over a finite field $\mathbb{F}_q$ are sufficient to achieve optimal flow rates; this approach is called random linear network coding (RLNC). With RLNC, linear combinations are built by multiplying the n original messages $m^{(1)}, m^{(2)}, \ldots, m^{(n)}$ with the random coefficients, the j-th linear combination $X^{(j)}$ being:

$$\boldsymbol{X}^{(j)} = \sum_{i=1}^{n} c_i^{(j)} \boldsymbol{m}^{(i)}. \tag{1}$$

When multiple such combinations are received, they form a system of linear equations. The original messages can be retrieved by solving the system with, for example, Gaussian elimination (GE), once sufficient combinations have been received. In general, it is not possible to decode a subset of messages with fewer than n linear combinations. However, all messages can be decoded immediately once enough linear combinations were received. This has been described as the "all-or-nothing property" [15].

As Nguyen *et al.* [11] note, for many applications, NC's delay is not tolerable. Consequently, Nguyen *et al.* propose PNC, which is based on RLNC and reduces per-packet delay. PNC introduces hierarchical layers $R_1, R_2, \ldots, R_{|\boldsymbol{R}|}$ of prioritized packets. That is, linear combinations of the l-th layer encode only messages from $m^{(1)}, m^{(2)}, \ldots, m^{(R_l)}$:

$$\boldsymbol{x}^{(j)} = \sum_{i=1}^{R_l} c_i^{(j)} \boldsymbol{m}^{(i)}, \quad \text{for a layer-}l\text{ combination.} \tag{2}$$

An important question is how to determine which layer to choose for generating new linear combinations. The most basic approach, used by HNC [11], is to choose layers uniformly at random. HNC generally provides lower per-packet delay than RLNC, but increases the overhead due to non-informative linear combinations, i. e., linear combinations from layers that can already be decoded.

Esmaeilzadeh *et al.* [14] explicitly studied the layer selection problem both for systems without any knowledge about the receivers' decoder states and for systems with perfect knowledge about the decoder states. The proposed layer selection algorithm is based on an exhaustive search through packet erasures. In addition, finite-horizon Markov decision processes are proposed for the perfect-knowledge system model. The authors describe their perfect-knowledge model as idealistic, since perfect knowledge is usually unavailable. Additionally, both algorithms' high computational complexity makes them unusable for practical applications with greater numbers of layers and/or users, but they may serve as a theoretical upper bound. We, different to [14], assume *limited* knowledge of the neighbors' decoder states and derive a simpler performance indicator that does not require exhaustive searching.

Naumann *et al.* [13] denote that all efficient layer selection schemes for PNC will send linear combinations roughly in order of priority, which they exploit to implement specialized Gaussian-elimination-based decoders. Such decoders improve both memory footprint and computational decoding complexity by reordering rows and inverting certain GE elimination steps in a joint decoder matrix for all layers. Our approach is a natural fit for such decoders, as it is based on a greedy strategy that sends in order of prioritization most of the time. Even more so, we provide an upper bound on the limit of deviation from this greedy strategy so that the asymptotic bounds on computational decoding overhead described in [13] hold.

Shenglan Huang *et al.* [16] build upon HNC with uniform random layer selection to minimize the amount of redundant packets sent in a multi-sender use case. Their algorithm estimates, according to loss rates and information about links, the ideal number of linear combinations that each sender should produce to reduce linearly dependent combinations. The algorithm does not, however, provide layer selection capabilities different from HNC.

Chau *et al.* [17] also use the HNC coding technique but propose an additional coding scheme that combines messages from more than one HNC-coded generation. Thereby, the scheme provides additional redundancy, which protects against packet loss, and reduces the number of transmissions until all layers can be decoded. Our proposed layer selection technique could be used in conjunction with their HNC-based coding scheme, since it does not modify the coding format.

Approaches different from hierarchical PNC have been proposed to reduce per-packet delay in NC: Shrader *et al.* [18], for example, propose to employ a systematic coding approach. Namely, a subset of the network's nodes sends uncoded packets in some circumstances. Due to the selection of nodes, the level of error protection is not reduced, but the non-encoded packets reduce per-packet delay as they do not require decoding. Yan *et al.* [15] instead trade correctness of decoded information for an increased chance of rank-deficient decoding by regarding the decoder matrix as a collection of underdetermined systems and implementing rank-deficient decoders. Finally, Claridge *et al.* [19] demonstrate that rank deficient decoding without chance of error is feasible when using a small enough finite field. Such a small field, however, also reduces the level of error protection and increases the chance of linear dependency.

III. System Model

A. Information model

We assume that the information to be transmitted by a source node can be split into equally sized messages $\boldsymbol{M} = (\boldsymbol{m^{(1)}}, \boldsymbol{m^{(2)}}, \ldots, \boldsymbol{m^{(n)}})$. Each message $\boldsymbol{m^{(i)}}$ consists of b symbols $(m_1^{(i)}, m_2^{(i)}, \ldots, m_b^{(i)})$ over a finite field $\mathbb{F}_q$. We are concerned with prioritized messages, i. e., some messages convey more information than others. In the following, and

w. l. o. g., we assume $\boldsymbol{m}^{(i)}$ has higher priority than $\boldsymbol{m}^{(j)}$ for all $1 \leq i < j \leq n$. PNC introduces hierarchical layers that we formalize as the vector $\boldsymbol{r} = (r_1, r_2, \ldots, r_{|\boldsymbol{r}|}) \in \mathbb{N}^{|\boldsymbol{r}|}$. Each entry r_l denotes the number of messages that the layer l adds, so it holds that $n = \sum_{l=1}^{|\boldsymbol{r}|} r_l$. Analogously, we define $\boldsymbol{R}$ as the cumulated layer vector with $R_i = \sum_{l=1}^{i} r_l$.

To transmit information, each source creates a sequence of network-coded packets; the j-th packet has the form $(\boldsymbol{c}^{(j)}, \boldsymbol{x}^{(j)})$. Here, $\boldsymbol{c}^{(j)}$ is called the encoding vector and consists of randomly chosen coefficients $(c_1^{(j)}, c_2^{(j)}, \ldots, c_n^{(j)})$, i. e., random symbols over $\mathbb{F}_q$. The second component, $\boldsymbol{x}^{(j)}$, is called the information vector and contains the actual linear combination. An information vector of the l-th ($1 \leq l \leq |\boldsymbol{r}|$) layer and j-th coded packet is encoded as follows [1], [12]:

$$x_k^{(j)} = \sum_{i=1}^{R_l} c_i^{(j)} m_k^{(i)} \quad \forall 1 \leq k \leq b \tag{3}$$

Since finite field operations are generally performed over all symbols of a message or information vector, we usually omit the symbol index k, which reduces Equation (3) to Equation (2). Multiple generations or multiple source nodes fit the above definitions by executing the encoding mechanism repeatedly in sequence or in parallel, respectively.

The finite field $\mathbb{F}_{2^8}$ is a typical choice for network coding [1], [20] and is used in the following. With $\mathbb{F}_{2^8}$, linear dependencies between randomly generated combinations are unlikely [9], and the elements' binary representations occupy one byte each, which is advantageous in practical systems.

B. Network model

Our network model is a wireless network with multiple nodes. Nodes transmit information as broadcast messages. Such messages may be received or lost by multiple nodes independently. We allow several nodes to be source nodes, which generate new messages $\boldsymbol{M}$. Both source nodes and non-source nodes receive, re-encode, and transmit information.

Before any transmission starts, a source node holds the complete data set, which is to be transmitted via PNC to the sink nodes. To simplify the system model, we assume in this work that all nodes in the network are sink nodes, which is a typical simplification for network coding. It alleviates the need to share topology information and improves the network capacity utilization [21]. The generality of our results is not affected by this simplification: if all nodes within the network receive the data, then any single sink or few sinks trivially have the information, too. As a consequence, a source node of one transmission is a sink node to other nodes, as well.

C. Problem statement

According to our information model, data to be transmitted is partitioned into different priority layers. Given this partition, the problem statement is to find an *efficient* transmission process that implements the prioritization scheme that is defined by the layers. Here, "efficient" comprises two aspects: low per-layer delay and low number of total transmissions. The per-layer delay for prioritized layers describes the prioritization performance, whereas the total number of transmissions describes the overhead that is introduced. Ideally, both aspects are jointly optimized by *implementing effective prioritization without introducing overhead.*

It is impossible, however, to achieve low delay at the same time as low overhead in all situations, as these goals may conflict. Rather, we propose a protocol that jointly optimizes both whenever possible and prioritizes low delay in conflict situations. To understand the connection between delay and total transmission number, consider an example topology with one source node S and three non-source nodes N_1, N_2, and N_3. Assume a simple PNC system with a generation size $n = 4$ and two priority layers $\boldsymbol{r} = (2, 2)$. We will now discuss two scenarios: one where delay and overhead are in conflict and one where both can jointly be optimized.

As the first scenario, assume nodes N_1, N_2, and N_3 have received $1, 1$, and 0 linearly independent combinations of the first layer, respectively, and only N_3 has received 1 linear combination of the second layer. Now, S sends two more linear combinations of the first layer. Then, N_1 and N_2 can decode the first layer after having received the first linear combination, and node N_3 can decode after having received both linear combinations. However, two more linear combinations of the second layer must be sent before all nodes can retrieve the second priority layer. If, instead, S immediately starts sending linear combinations of the second layer, all nodes must wait for one more transmission before they can decode the first layer. After only three transmissions in total (instead of four, as before), all nodes can decode layer one *and* two.

As the second scenario, consider nodes N_1, N_2, and N_3 have initially received $2, 1$, and 0 independent combinations of the first layer and $1, 2, 3$ independent linear combinations of the second layer, respectively. In this case, sending only one combination of the (lower priority) second layer from the beginning saves one transmission *and* achieves minimal per-layer delay.

The important observation here is that the goals of prioritization and not introducing extraneous transmissions *can* conflict, but this is *not always* the case. Our goal is to provide optimal prioritization first, but identify such occasions where we can save transmissions without introducing per-layer delay.

IV. Proposed Algorithm

In this section, we approach the problem statement with a simplified, theoretical model: nodes have knowledge of their neighbors' decoder matrix rank and each layer's linear subspace dimension. There are no packet losses or delays. We describe the proposed algorithm, iNsPECt, from the perspective of an individual node. In Section V, we incorporate packet losses, delays, and the need for feedback messages in the design of a network protocol based on this algorithm.

Our proposed algorithm combines two complementary strategies, which we term "Ord" and "SL," that are used to decide which layer to select for transmission of the next linear combination. Both strategies (and our algorithm) make use of the fact that when one node's decoder matrix for a given layer has a higher rank than its neighbor's, sending a linear combination is with high probability innovative. "Ord," short for *in order,* is a greedy strategy that selects layers in strict order of prioritization; "SL" sends linear combinations of a *single layer* only. Our key idea is to use strategy Ord by default to minimize delay for prioritized layers. But we resort to sending a lower priority layer (with strategy SL) if it reduces the required total transmissions and does not negatively affect per-layer delay. Strategy SL takes a target layer i as a parameter; $\mathrm{SL}(i)$ sends only linear combinations of layer i until layer i – and thus all higher priority layers, as well – can be decoded. Obviously, $\mathrm{SL}(i)$ requires the minimum number of transmissions until layer i can be decoded; and strategy $\mathrm{SL}(|\boldsymbol{r}|)$ equals RLNC. Strategy Ord instead sends linear combinations of the highest priority non-decodable layer until each neighbor can decode that layer. It then continues with the next layer. Therefore, Ord ensures that high priority layers are always decoded before lower-priority layers.

To determine which strategy to use, our algorithm models the benefits of choosing one strategy over the other at any point in time with two performance indicators. Each indicator takes a parameter i and returns the benefits or drawbacks that result from choosing strategy $\mathrm{SL}(i)$ over Ord.

The first indicator, $Q_{\mathrm{rt}}(i)$, counts the *savings in total number of transmissions* until each neighbor of a node can decode layer i. Positive values indicate that using $\mathrm{SL}(i)$ is beneficial over choosing Ord. The second indicator, $Q_{\mathrm{dc}}(i)$, analogously counts the *additional per-layer delay* (in transmissions) until a node's neighbors can decode layers 1 to i, cumulated over the layers and all neighbors. Positive values indicate additional overhead introduced by choosing $\mathrm{SL}(i)$. In the following, we first derive the two indicators from our simplified system model and then define the selection algorithm.

A. Performance indicators

1) Reduction in transmissions: We define $\mathrm{RT}_{\mathrm{SL}}^{(*)}(i)$ as the number of required transmissions until all neighbors can decode layer i using the SL strategy. Analogously, $\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i)$ denotes the number of required transmissions using the Ord strategy. Consequently, the savings in transmissions are:

$$Q_{\mathrm{rt}}(i) = \mathrm{RT}_{\mathrm{Ord}}^{(*)}(i) - \mathrm{RT}_{\mathrm{SL}}^{(*)}(i). \tag{4}$$

Next, we derive $\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i)$ and $\mathrm{RT}_{\mathrm{SL}}^{(*)}(i)$. Let $\gamma_l^{(x)}$ be the number of independent linear combinations of layer l that neighbor x has received (and equivalently, the number of dimensions of the linear subspace that pertains to layer l), and let $\Gamma_l^{(x)}$ be the accumulated number of received combinations from layer 1 to l of neighbor x. Let $\mathrm{RT}_{\mathrm{SL}}^{(x)}(i)$ be the number of transmissions required for a single node x to decode layer i. Layer i can be decoded once layer i's linear subspace has full rank, i. e., when $\Gamma_i^{(x)} = R_i$. Alternatively, layer i may be decoded when a lower priority subspace has full rank, i. e., $\Gamma_j^{(x)} = R_j$ for some $j > i$. Thus,

$$\mathrm{RT}_{\mathrm{SL}}^{(x)}(i) = \min\left(R_i - \Gamma_i^{(x)}, \min_{j=i+1}^{|\boldsymbol{r}|}\left(R_j - \Gamma_j^{(x)}\right)\right)$$

$$\Leftrightarrow \mathrm{RT}_{\mathrm{SL}}^{(x)}(i) = \min_{j=i}^{|\boldsymbol{r}|}\left(R_j - \Gamma_j^{(x)}\right). \tag{5}$$

To generalize $\mathrm{RT}_{\mathrm{SL}}^{(x)}(i)$ to $\mathrm{RT}_{\mathrm{SL}}^{(*)}(i)$, we take the maximum over all neighbors:

$$\mathrm{RT}_{\mathrm{SL}}^{(*)}(i) = \max_{x\,\in\,\text{Neigh.}} \mathrm{RT}_{\mathrm{SL}}^{(x)}(i). \tag{6}$$

We construct $\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i)$ recursively. Since strategies Ord and SL are identical for $i = 1$, it holds that

$$\begin{aligned} &\mathrm{RT}_{\mathrm{Ord}}^{(x)}(1) = \mathrm{RT}_{\mathrm{SL}}^{(x)}(1), \text{ and} \\ &\mathrm{RT}_{\mathrm{Ord}}^{(*)}(1) = \max_{x\,\in\,\text{Neigh.}} \mathrm{RT}_{\mathrm{SL}}^{(x)}(1) = \mathrm{RT}_{\mathrm{SL}}^{(*)}(1). \end{aligned} \tag{7}$$

Counting the required transmissions for the $(i+1)$-th layer, we first count the transmissions from the i-th layer and then add the remaining, maximum missing matrix rank over all neighbor nodes:

$$\begin{aligned} &\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i+1) = \\ &\qquad \mathrm{RT}_{\mathrm{Ord}}^{(*)}(i) + \max_{x\,\in\,\text{Neigh.}}\left(\mathrm{RT}_{\mathrm{SL}}^{(x)}(i+1) - \mathrm{RT}_{\mathrm{SL}}^{(x)}(i)\right), \end{aligned} \tag{8}$$

which, if we define $\mathrm{RT}_{\mathrm{SL}}^{(x)}(0) = 0$, reduces to

$$\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i) = \sum_{j=0}^{i-1} \max_{x\,\in\,\text{Neigh.}}\left(\mathrm{RT}_{\mathrm{SL}}^{(x)}(j+1) - \mathrm{RT}_{\mathrm{SL}}^{(x)}(j)\right). \tag{9}$$

2) Per-layer delay: Analogously, we define $\mathrm{DC}_{\mathrm{SL}}^{(x)}(i)$ and $\mathrm{DC}_{\mathrm{Ord}}^{(x)}(i)$ as the cumulative per-layer delay (in transmissions) until node x can decode each layer up to i with the SL and Ord strategies, respectively. Similarly, $\mathrm{DC}_{\mathrm{SL}}^{(*)}(i)$ and $\mathrm{DC}_{\mathrm{Ord}}^{(*)}(i)$ define this delay cumulatively for all neighbor nodes. Our indicator,

$$Q_{\mathrm{dc}}(i) = \mathrm{DC}_{\mathrm{SL}}^{(*)}(i) - \mathrm{DC}_{\mathrm{Ord}}^{(*)}(i), \tag{10}$$

gives the additional per-layer delay that results from choosing strategy SL over Ord.

With the SL strategy, each node waits a timespan that is independent from the other nodes' decoder matrix states, as each node's rank of layer i increases independently until full rank is obtained. As the SL strategy only sends linear combinations of layer i, each non-decodable layer below i of neighbor x will become decodable after exactly $\mathrm{RT}_{\mathrm{SL}}^{(x)}(i)$ transmissions. Therefore, we count the number of non-decodable layers (right-hand factor) multiplied by the number of transmissions required for decoding each layer (left-hand factor):

$$\mathrm{DC}_{\mathrm{SL}}^{(x)}(i) = \mathrm{RT}_{\mathrm{SL}}^{(x)}(i) \cdot \sum_{k=1}^{i} \min\left(\mathrm{RT}_{\mathrm{SL}}^{(x)}(k), 1\right), \tag{11}$$

$$\mathrm{DC}_{\mathrm{SL}}^{(*)}(i) = \sum_{x\,\in\,\text{Neigh.}} \mathrm{DC}_{\mathrm{SL}}^{(x)}(i). \tag{12}$$

Again, we derive the per-layer delay for the Ord strategy recursively and in two steps. First, we derive the non cumulated per-layer delay $\mathrm{NC}_{\mathrm{Ord}}^{(x)}(i)$ so that

$$\mathrm{DC}_{\mathrm{Ord}}^{(x)}(i) = \sum_{j=1}^{i} \mathrm{NC}_{\mathrm{Ord}}^{(x)}(j). \tag{13}$$

Again, for $i = 1$ both strategies behave identically, thus $\text{DC}^{(x)}_{\text{Ord}}(1) = \text{NC}^{(x)}_{\text{Ord}}(1) = \text{DC}^{(x)}_{\text{SL}}(1)$ and $\text{DC}^{(*)}_{\text{Ord}}(1) = \text{DC}^{(*)}_{\text{SL}}(1)$. For $i+1$, we distinguish between two cases based on whether the $(i+1)$-th layer can be decoded if the i-th layer can be decoded, which formally is the proposition:

$$\text{RT}^{(x)}_{\text{SL}}(i+1) = \text{RT}^{(x)}_{\text{SL}}(i). \tag{14}$$

If eq. (14) holds, the induction step is trivial, as no additional delay comes from layer $i+1$: $\text{NC}^{(x)}_{\text{Ord}}(i+1) = \text{NC}^{(x)}_{\text{Ord}}(i)$. If eq. (14) does not hold, node x requires exactly as many independent linear combinations as it is short of full rank to decode layer $i+1$, i.e., $\text{RT}^{(x)}_{\text{SL}}(i+1) - \text{RT}^{(x)}_{\text{SL}}(i)$. Since the Ord strategy will not start sending linear combinations of rank $i+1$ until *all* other neighbors can decode layer i, we also have to wait for $\text{RT}^{(*)}_{\text{Ord}}(i)$ transmissions before the $(i+1)$-th layer's rank increases:

$$\text{NC}^{(x)}_{\text{Ord}}(i+1) = \begin{cases} \text{NC}^{(x)}_{\text{Ord}}(i), & \text{if eq. (14) holds, else} \\ \text{RT}^{(*)}_{\text{Ord}}(i) + \text{RT}^{(x)}_{\text{SL}}(i+1) - \text{RT}^{(x)}_{\text{SL}}(i). \end{cases} \tag{15}$$

For all nodes, we obtain:

$$\text{DC}^{(*)}_{\text{Ord}}(i) = \sum_{x \,\in\, \text{Neigh.}} \text{DC}^{(x)}_{\text{Ord}}(i). \tag{16}$$

B. Algorithm

We have defined the two performance indicators $Q_{\text{rt}}(i)$, which counts the required transmissions until layer i can be decoded by all neighbor nodes, and $Q_{\text{dc}}(i)$, which counts the per-layer delay over all neighbors and layers up to i. We now use these indicators in an algorithm that takes a node's state as input and returns the layer choice as output. Whenever a node generates and transmits a linear combination, it executes the algorithm iNsPECt first, which is given in Figure 1.

To keep computational overhead low in practical systems, iNsPECt introduces a system parameter k_{ahead}, which bounds the number of layers that a node may deviate from the Ord strategy. For large numbers of layers, bounding the deviation with k_{ahead} not only improves our algorithm's performance, but allows to use optimized decoding techniques [13].

In Figure 1 line 1, the Ord strategy is employed by default. That is, a node selects the layer with the highest priority that any of its neighbors cannot decode. To reduce the total number of transmissions, our algorithm uses the previously defined performance indicators in two steps:

(1) check if a lower priority layer can save transmissions by computing Q_{rt} and
(2) check if the lower priority negatively affects per-layer delay by computing Q_{dc}.

These steps are repeatedly performed in the conditional at line 4 for all k_{ahead} candidate layers in the loop at lines 3 to 8. Whenever a candidate layer in the loop further reduces transmissions compared to the last candidate *and* does not increase per-layer delay, it is selected as next candidate. Finally, the selected layer is returned in line 9.

```
Input: for each neighbor $x \in$ Neigh., $\boldsymbol{\gamma}^{(x)}$ and $\boldsymbol{\Gamma}^{(x)}$
Output: layer index
Have: $n, \boldsymbol{r}, \boldsymbol{R}$, and system parameter $k_{ahead}$
1: choice $\leftarrow$ start $\leftarrow$ first non decodable layer of Neigh.
2: last$_{\text{rt}} \leftarrow Q_{\text{rt}}(start)$
3: for $i \leftarrow \text{start}+1, \dots, \min(\text{start}+k_{\text{ahead}}, |\boldsymbol{r}|)$ do
4:     if $Q_{\text{rt}}(i) > \text{last}_{\text{rt}} \wedge Q_{\text{dc}}(i) \le 0$ then
5:         choice $\leftarrow i$
6:         last$_{\text{rt}} \leftarrow Q_{\text{rt}}(i)$
7:     end if
8: end for
9: return choice
```

Fig. 1. iNsPECt.

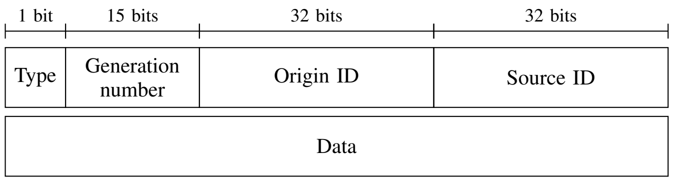


Fig. 2. Message format: general header and message specific data part.

Dependent on m, the number of neighbors from which feedback has recently been received, the algorithm's runtime is in $\mathcal{O}(m\, k_{\text{ahead}}\, |\boldsymbol{r}|)$. For three layers, a common choice in multimedia streaming [11], the runtime is linear in the number of neighbors.

V. Protocol Design

We now describe a simple yet effective network protocol that instantiates the layer selection algorithm for practical systems. We describe the protocol for a single network coding generation and a single source. The protocol is executed in parallel when multiple sources exist in the network, and it is run repeatedly for subsequent generations.

A. Message types

The protocol requires only two types of messages: data messages, which contain linear combinations, and feedback messages, which concisely encode a node's decoder state. We use UDP as the underlying transport protocol, because reliability is ensured by network coding's forward error correction properties.

The general message format is shown in Figure 2: a leading bit is used to denote message type, and the remaining 15 bits of the first two octets encode the generation number that the message belongs to. Origin node and source node, each encoded using 32 bits, are used to manage neighbor state and assign linear combinations to the correct decoding system. Each node is assigned a unique number in the system; alternatively, the IP addresses can be used for local topologies.

Data messages contain linear combinations, which consist of an encoding vector and an information vector, as described in Section III-A. If a linear combination from a layer with higher priority is sent, not all coefficients in the encoding vector are used; in that case, the remaining coefficients are set to the additive identity (i. e., “zeroes”) of the finite field. Thus, encoding vectors contain n coefficients, whereas the information vector consists of b symbols. Each coefficient and symbol are elements of the finite field F_{2^8} and can be encoded as a single byte. Data messages, therefore, have a length of $b + n + 10$ bytes.

Feedback messages encode $\boldsymbol{\gamma}^{(x)}$; that is, they encode how many linearly independent combinations of each layer a node x has received. This is identical to the rank of each layer’s decoder matrix or the dimension of each layer’s linear subspace. A feedback message encodes each rank with two bytes; thus, the total feedback message length is $2 \cdot |\boldsymbol{r}| + 10$ bytes, where $|\boldsymbol{r}|$ is the total number of layers. Since the feedback messages’ size does not depend on the generation size n nor on the chunk size b, feedback messages are usually much smaller than data messages.

B. Transmission mechanism

We employ a constant-rate approach to sending linear combinations. That is, every data message transmission interval λ_{data}, a linear combination is built according to Equation (3), encoded as a data message, and sent when the generation is not marked as fully transmitted. First, the iNsPECt algorithm is executed to determine the ideal layer i for building the next linear combination. If that layer-i’s decoder matrix has insufficient rank to generate a linear combination, i is incremented repeatedly until a linear combination can be built. If, initially, no feedback is available, we default to sending a linear combination of the highest priority layer.

Whenever a layer-i linear combination is sent, the feedback vector $\boldsymbol{\gamma}^{(x)}$ for each neighbor x is incremented, presuming the linear combination’s successful reception. In addition, a flag is set that indicates that the feedback vector is *assumed* instead of authoritative. If an assumed feedback vector indicates the generation is fully transmitted, the node continues to send linear combinations until an authoritative feedback message is received as confirmation. Thereby, nodes avoid delays at the end of each generation. By handling assumed and authoritative feedback in this way, we ensure that layers of lower-than-ideal priority may be selected, but never layers of higher priority. This bias may lead to increased per-layer delay for the current layer. It does not, however, cause additional overhead from linearly dependent combinations, nor does it affect the lower priority layers’ decoding delay.

On reception of a linear combination, a node first counts the trailing number of additive identity elements in the encoding vector to determine the combination’s layer. Next, the linear combination is inserted into each decoder matrix that pertains to a lower or equal priority than the linear combination’s layer. Whether the newly received linear combination is innovative is determined using GE. If the rank of the matrix increases, the combination was innovative; otherwise, the new row is reduced to additive identity elements [1].

C. Feedback mechanism

Feedback is broadcast periodically and cumulatively for a generation’s layers: once every feedback interval λ_{fb}, a feedback message is created for each incomplete generation. Usually, feedback messages are only sent when the generation is incomplete, i. e., the node’s decoder matrix state does not have full rank for all layers. If, however, a linear combination for a complete generation is received, a feedback message that indicates successful reception of the whole generation (with $\Gamma_{|\boldsymbol{r}|} = R_{|\boldsymbol{r}|}$) is sent once.

The feedback’s purpose is not only to inform other nodes about the current decoder state, but it also allows other nodes to learn about their neighborhood. When a node receives a feedback message from a node x, it includes x in its neighbor set (Neigh. in Figure 1) and updates $\boldsymbol{\gamma}^{(x)}$ with the rank vector that is included in the feedback message. If a feedback message is received where the combination of origin identifier and generation number is unknown, that message is ignored, as the feedback contained is not helpful. Feedback vectors and neighborhood states expire after a timeout that should be chosen as a multiple of the feedback interval to avoid incomplete neighbor sets.

The proportion between the data interval λ_{data} and λ_{fb} is important for the performance of the proposed algorithm. The smaller the feedback interval, the better each node’s stored feedback represents its neighbors’ decoder state, since it is updated more often. On the other hand, even tough feedback messages have a small size, more frequent feedback means more network capacity is used for traffic that does not directly contribute to the delivery of the sources’ information.

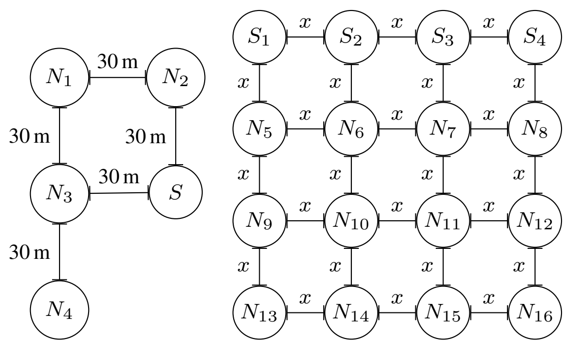
VI. Evaluation

We compare iNsPECt to PNC’s HNC variant and RLNC, which we both described in Section II. As a *lower bound* on decoding delay, we additionally show the decoding time of the first layer, which results from sending only linear combinations of the first layer. We evaluate all protocols in three scenarios: a small-scale topology with a single source and small generations that comprise few source messages to evaluate the impact of varying feedback rates, a larger topology to show multi-hop capabilities and support for multiple sources, and finally a set of larger, randomized topologies to support the generality of our results.

A. Methodology

We evaluate using the discrete event network simulator ns-3 (version 3.25) [22]. Wireless links between nodes are modeled via YANS Wifi model [23] with 802.11g MAC and 2.4 GHz PHY. We simulate the physical channel with the log-distance propagation loss model[1] and the Rayleigh fast fading model, one superimposed on the other [24]. Nodes send actual linear combinations in the simulation, so there is a (small) chance for linear dependency even if a layer’s decoder matrix does

[1]We choose the path-loss exponent $\gamma = 3.0$ and configured path loss at the reference distance 1 m according to Friis’ model for 2.4 GHz, which is in line with a range of office and industrial environments [24], [25].



(a) Small scale topology (b) Larger network topology

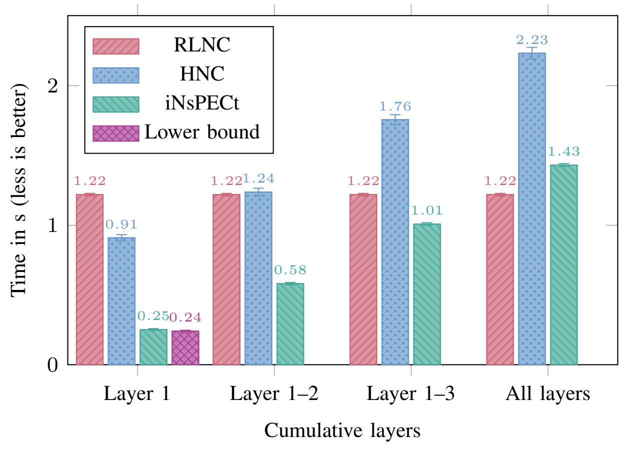
Fig. 3. Simulated topologies.

not have full rank. Each simulation is run for 200 s simulated time and executed 5 times. Each repeated simulation uses a separate sub-stream of ns-3's MRG32k3a pseudo-random number generator to ensure uncorrelated results [26]. Pseudo-randomness is used in the simulation's wireless fading model, the ns-3 bit error model, which is affected by fading and path loss, generation of NC coefficients, and exponentially distributed variations in packet send times that we use to avoid collisions and other synchronization effects. Since generations take much less than 200 s to transmit, hundreds of transmitted generations contribute to a statistically meaningful sample size. All figures in this section show the sample mean and *95% confidence intervals* (assuming normal distribution). Error bars may not be visible when the confidence is very high.

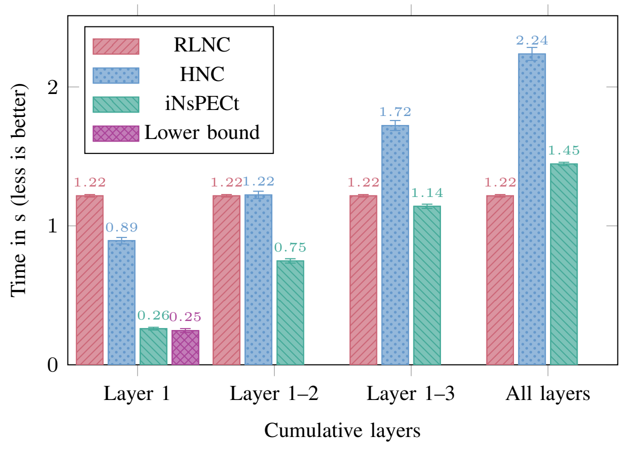
B. Small-scale topology

Figure 3a shows the small-scale topology: one source S and four nodes N_1 to N_4 are arranged in a partial grid with 30 m grid width. Due to the grid layout, nodes N_1 and N_2 have 30 m distance from the center-positioned source, whereas nodes N_2 and N_4 are 42 m away, which results in lower packet delivery probability (PDR) for these nodes. We evaluate a small generation ($n = 10$) with small layers $\boldsymbol{r} = (2, 2, 3, 3)$ to highlight the effects of the layer selection and feedback rates.

The mean time until all sink nodes are able to decode layers 1 to 4, respectively, is given in Figure 4 for two different feedback rates. Results for frequent feedback are shown in Figure 4a: the y-axis shows the average time from beginning to transmit the current generation until a layer can be decoded by a node. The x-axis gives the individual layers, which are inherently cumulative due to the hierarchical nature of layers. It can be seen that using RLNC, the time until all layers can be decoded is identical for all layers. Since RLNC offers maximum protection against erasures and has the lowest chance to send linearly dependent combinations, it gives us the optimal decoding time for layer 4 (and thus all layers) in the last column. The HNC strategy, being fully randomized and independent of any feedback, provides faster recovery of the highest prioritized first layer, identical decoding time for the second layer, and significantly worse delay than RLNC for



(a) *High* feedback rate ($\lambda_{\text{data}/\lambda_{\text{fb}}} = 1/1$)

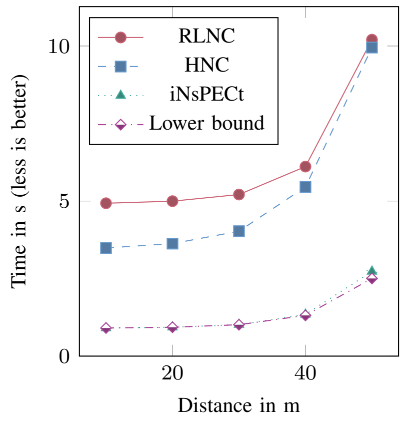


(b) *Low* feedback rate ($\lambda_{\text{data}/\lambda_{\text{fb}}} = 1/3$)

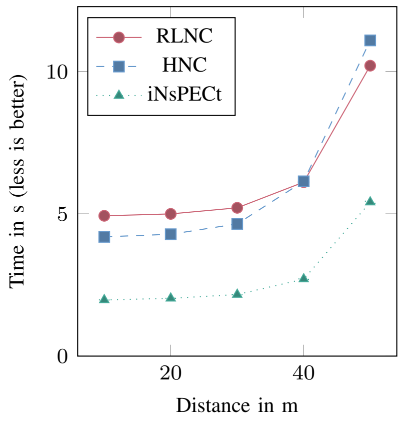
Fig. 4. Small topology results: per-layer delay for different feedback rates.

the third and fourth layer. The proposed algorithm, iNsPECt, consistently outperforms HNC between 35.9 % and 72.3 %. Also, the delay is just 4.5 % higher than the *lower bound* for the most highly prioritized 1st layer (compared to 277.5 % for HNC). The layer 4 decoding delay of iNsPECt is only 17.3 % higher than the optimal RLNC delay. These 17.3 % analogously give the overhead imposed by the prioritization scheme, since the additional delay corresponds to the number of linearly dependent combinations that are due to prioritization. In comparison, this overhead is 83.1 % for HNC.

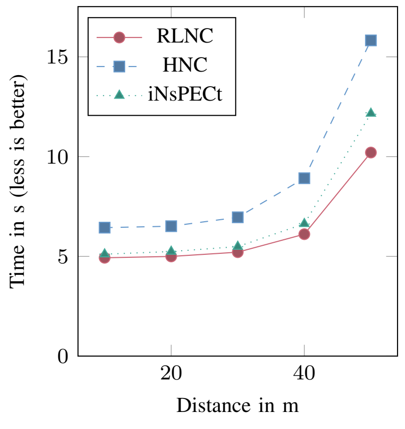
In a second step, we lowered the feedback rate to $1/3 \cdot \lambda_{\text{data}}$, which is quite low compared to the individual layer sizes: without losses, another layer has to be selected every two or three linear combinations or all subsequently sent combinations are linearly dependent. The results are shown in Figure 4b in a format identical to Figure 4a: RLNC and HNC, working without feedback, are unaffected by the change. iNsPECt is still faster than HNC for all layers, but due to the lack of recent feedback and the resulting uncertainty about the neighbors' decoder states, the algorithm resorts to sending layers of lower priority than necessary, which have a much lower chance of



(a) Layer 1



(b) Layers 1–2



(c) All layers (1–4)

Fig. 5. *Larger* topology results: per-layer delay for different (cumulative) layers and varying distances.

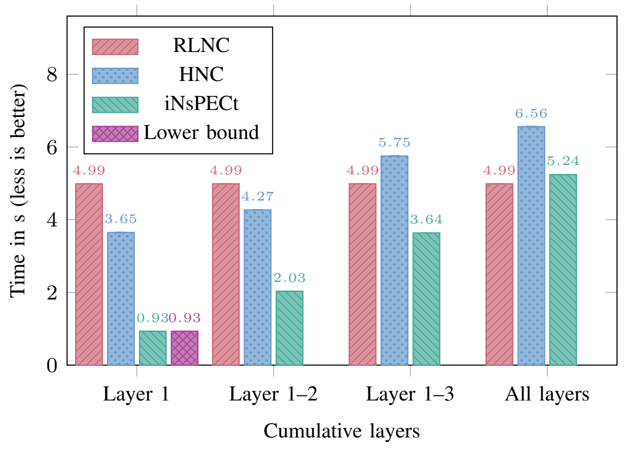


Fig. 6. Random topology results: average per-layer delay.

linear dependency. The downside of this approach can best be seen in the second and third layers, where decoding delay is still 38.8 % and 33.8 % better than HNC, but also increases by 28.6 % and 13.1 % compared to the better feedback rate scenario. A positive aspect of resorting to lower priority linear combinations is that linear dependency is less likely, which can be seen best at the fourth layer, where decoding time is not significantly different from the former scenario. This means that albeit iNsPECt's per-layer delay is not as low as before, it is still better than HNC *and* total prioritization overhead compared to RLNC does not increase at all.

C. Larger topology and random topology

We now demonstrate scalability to larger ad-hoc networks with multi-hop requirements. The larger network topology is shown in Figure 3b and has 16 nodes in total: 4 source nodes and 12 non-source nodes. As before, nodes are arranged in a grid, but now we have four nodes in each row. We simulate a larger generation with $n = 50$ and 4 layers with layer sizes $\boldsymbol{r} = (10, 10, 15, 15)$. Figure 5 shows the simulation results for medium feedback rate ($\lambda_{\text{data}/\lambda_{\text{fb}}} = 1/2$) and varying grid distances x between neighbor nodes from 10 m to 50 m. Figures 5a to 5c give decoding delay for layer 1, layers 1–2, and layers 1–4, respectively.

iNsPECt allows nodes to retrieve the most highly prioritized 1st layer 74.1 %, 74.7 %, and 72.6 % faster than HNC for node distances of 10 m, 30 m, and 50 m, respectively. Also, the first layer's retrieval time with iNsPECt almost matches the lower bound; only at 40 m and 50 m distance, the results show 2.4 % and 9.0 % delay for the 1st layer.

Results look similar for the second layer in Figure 5b, where iNsPECt yields a 52.9 % to 51.2 % reduced per-layer delay over HNC. Interestingly, the benefit of HNC over RLNC for prioritized layers diminishes with greater distances between nodes (and thus lower PDR): at 40 m distance between nodes, HNC gives roughly the same per-layer delay as RLNC. We attribute the poor performance of HNC with low PDR in the large-scale scenario to inefficient multi-hop capabilities: when the source has few opportunities to successfully transmit linear combinations to the inner nodes in the network, low priority linear combinations are more useful, because they have much higher chance of being innovative.

As expected, Figure 5c shows that the last layer – and thus all layers due to the hierarchical layer structure – is decoded the fastest with RLNC, which has the highest level of error protection against packet loss and the lowest chance of sending linear combinations of layers that already have full rank in neighbors. The delay given in Figure 5c is a direct indicator for the total number of transmissions required for sending one full generation. Therefore, it is also indicative for achievable throughput. For distances at or below 30 m, iNsPECt has at most 5.4 % overhead compared the optimum RLNC. This overhead increases to 8.5 % at 40 m distance between neighbors and 19.2 % at 50 m. HNC, in comparison, results in a message overhead of 55.1 % over RLNC.

Last, we verify the system's properties in a set of larger randomized topologies. Figure 6 shows mean per-layer delay for twenty different topologies where the nodes' locations are selected uniform at random in a 90 m × 90 m square. Again, we see a better performance of iNsPECt than HNC for all layers and a low total overhead of only 5.1 % compared to RLNC. Notably, iNsPECt enables decoding the most highly prioritized first layer 74.5 % faster than HNC.

D. Summary

We evaluated iNsPECt in both smaller and larger scenarios and compared it to HNC and RLNC for different feedback rates, distances, and topologies. iNsPECt significantly outperforms HNC in every scenario that we tested. Having only sporadic feedback and thus outdated decoder state information has no effect on the system's overhead in terms of linear dependency, but it increases decoding delay for some layers, albeit keeping delay significantly lower than HNC. Remarkably, in all scenarios, that is, for small and large distances, for high and low feedback rates, and for the small and larger-scale scenarios, the most highly prioritized layer's decoding delay was within 10 % of the optimum. The overhead of iNsPECt compared to RLNC over all scenarios is consistently less than 20 %, which we consider a low cost for having prioritization.

VII. Conclusion and Future Work

We describe a protocol that addresses a principal problem of existing prioritized network coding protocols. Namely, we answer the question which layer to use for generating linear combinations. Towards this end, we propose a novel, distributed algorithm, iNsPECt, that leverages limited feedback containing each layer's subspace dimension. iNsPECt defines two performance indicators that enable it to deviate from a greedy strategy in order to reduce prioritization overhead without affecting prioritization performance. Our evaluation shows that the proposed algorithm consistently outperforms HNC and, under good network conditions, approaches the lower bound on required transmissions that is achieved by non-prioritized RLNC. In addition, the highest priority layer's decoding delay is nearly optimal in all scenarios. Our results demonstrate that (1) prioritized network coding can be realized with low overhead, and (2) that even small feedback messages are sufficient for effective prioritization in such systems.

A future research direction is to observe the algorithm's performance in a more sophisticated network protocol: it is conceivable that observed channel conditions and limited network topology information supplied by such a protocol could be utilized to better estimate neighbors' decoder state if recent feedback is unavailable. In particular, we would expect a reduced per-layer delay for mid-priority layers when expected packet-loss rates are incorporated into the layer selection process that is used for generating linear combinations.

Acknowledgments

We thank Sebastian Henningsen for helpful discussion. Also, we thank our anonymous reviewers for their constructive feedback.

References

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

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

[3] Baochun Li and Di Niu, "Random Network Coding in Peer-to-Peer Networks: From Theory to Practice," 2011.

[4] Christos Gkantsidis, John Miller, and Pablo Rodriguez, "Comprehensive view of a live network coding P2P system," presented at the Internet Measurement Conference, 2006.

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

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

[7] 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.

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

[9] T. Ho, R. Koetter, M. Medard, *et al.*, "The benefits of coding over routing in a randomized setting," 2003.

[10] O. Trullols-Cruces, J. M. Barcelo-Ordinas, and M. Fiore, "Exact Decoding Probability Under Random Linear Network Coding," Jan. 2011.

[11] 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.

[12] D. Vukobratović and V. Stanković, "Unequal error protection random linear coding for multimedia communications," in *Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop On*, IEEE, 2010.

[13] R. Naumann, S. Dietzel, and B. Scheuermann, "Best of both worlds: Prioritizing network coding without increased space complexity," in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, Nov. 2016.

[14] M. Esmaeilzadeh, P. Sadeghi, and N. Aboutorab, "Random Linear Network Coding for Wireless Layered Video Broadcast: General Design Methods for Adaptive Feedback-Free Transmission," Feb. 2017.

[15] Z. Yan, H. Xie, and B. W. Suter, "Rank deficient decoding of linear network coding," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013.

[16] Shenglan Huang, Michele Sanna, Ebroul Izquierdo, *et al.*, "Optimized scalable video transmission over P2P network with hierarchical network coding," presented at the ICIP, 2014.

[17] 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.

[18] B. Shrader and N. M. Jones, "Systematic wireless network coding," in *MILCOM 2009 - 2009 IEEE Military Communications Conference*, Oct. 2009.

[19] J. Claridge and I. Chatzigeorgiou, "Probability of Partially Decoding Network-Coded Messages," 2017.

[20] Y. Wu, P. Chou, K. Jain, *et al.*, "A comparison of network coding and tree packing," in *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium On*, IEEE, 2004.

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

[22] T. R. Henderson, M. Lacage, G. F. Riley, *et al.*, "Network simulations with the ns-3 simulator," 2008.

[23] M. Lacage and T. R. Henderson, "Yet another network simulator," in *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, ACM, 2006.

[24] H. Hashemi, "The indoor radio propagation channel," 1993.

[25] S. Phaiboon, "Space Diversity Path Loss in a Modern Factory at frequency of 2.4 GHz," 2014.

[26] P. L'Ecuyer, R. Simard, E. J. Chen, *et al.*, "An Object-Oriented Random-Number Package with Many Long Streams and Substreams," Dec. 2002.