Received August 14, 2017, accepted September 18, 2017, date of publication September 27, 2017,
date of current version October 25, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2757241

Caterpillar RLNC (CRLNC): A Practical Finite Sliding Window RLNC Approach

SIMON WUNDERLICH[1], FRANK GABRIEL[1], SREEKRISHNA PANDI[1], FRANK H. P. FITZEK[1], AND MARTIN REISSLEIN[2]

[1]5G Laboratory Germany, Deutsche Telekom Chair of Communication Networks, Faculty of Electrical and Computer Engineering, TU Dresden, 01062 Dresden, Germany
[2]School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287-5706, USA

Corresponding author: Frank H. P. Fitzek (frank.fitzek@tu-dresden.de)

This work was supported in part by Ericsson, in part by Deutsche Telekom, and in part by the Free State of Saxony through funds from the European Commission for the Atto3-D Project.

ABSTRACT Random linear network coding (RLNC) is a popular coding scheme for improving communication and content distribution over lossy channels. For packet streaming applications, such as video streaming and general IP packet streams, recent research has shown that sliding window RLNC approaches can reduce the in-order delay compared with block-based RLNC. However, existing sliding window RLNC approaches have prohibitive computational complexity or require feedback from the receivers to the sender. We introduce caterpillar RLNC (CRLNC), a practical finite sliding window RLNC approach that does not require feedback. CRLNC requires only simple modifications of the encoded packet structure and elementary pre-processing steps of the received coded packets before feeding the received coding coefficients and symbols into a standard block-based RLNC decoder. We demonstrate through extensive simulations that CRLNC achieves the reliability and low computational complexity of block-based RLNC, while achieving the low in-order delays of sliding window RLNC.

INDEX TERMS Computational complexity, delay, random linear network coding (RLNC), sliding window.

I. INTRODUCTION

Random Linear Network Coding (RLNC) [2]–[4] is an increasingly popular method for efficiently transferring data in complex, chaotic, or lossy communication networks and systems. RLNC is well suited for wireless networks [5]–[13], data storage systems [14]–[17], and content distribution systems [18]–[22]. RLNC became practical through the introduction of block based RLNC approaches, which are also referred to as generation based RLNC approaches [23]–[25]. Block based RLNC partitions a large message or long data stream into several blocks (generations), whereby each block consists of g successive source symbols. The block based RLNC encoder and decoder operate only on the symbols within a given block at a time. This block-by-block operation enables RLNC encoding and decoding with low computational complexity compared to operating on the entire message or data stream. Thus, block based RLNC enables efficient, practical RLNC encoding and decoding with currently available CPU capabilities and memory architectures. However, the block-by-block operation introduces relatively large delay for coding all source symbols in the generation together in so-called full vector block based RLNC [4], [25] or for waiting for the coded packets at the end of the generation for recovering dropped packets in so-called systematic block based RLNC [26]–[28]. Alternatively, block based RLNC strategies that reduce the introduced delay, e.g., through distributing the coded packets among the source symbols for fast recovery of dropped symbols [29], incur higher source symbol drop probabilities. The drop probabilities increase because a given coded packet protects on average fewer source symbols since the range of protected source symbols is limited to within the current generation.

Recent RLNC research has developed sliding window RLNC approaches to achieve low in-order delivery delay for the source symbols without compromising the reliability of source symbol delivery [30]. The in-order delay is an important metric for all forms of streaming applications, such as multimedia streams and regular IP packet streams. Sliding window RLNC does not operate on distinct blocks of source symbols; instead, sliding window RLNC operates on source symbols within an encoding window that slides over the sequence of source symbols. Existing sliding window RLNC requires feedback for advancing the tail (closing) end of the encoding window, see Section II-C for details.

Without feedback, the window grows very large. Thus, sliding window RLNC has prohibitive computational complexity and memory requirements in networks without feedback.

In this paper, we introduce a new RLNC approach, the Caterpillar RLNC (CRLNC) approach, that closes the gap between block based RLNC and sliding window RLNC in networks without feedback. Akin to the crawling locomotion of a caterpillar [31], CRLNC slides a finite encoding window covering w_e successive source symbols across a stream of source symbols. We introduce a novel CRLNC packet structure that contains signalling information for the accurate RLNC decoding at the receiver. The CRLNC receiver operates with a decoding window that covers w_d successive source symbols. The decoding window is advanced according to the signaling information in the arriving CRLNC packets. The CRLNC decoder conducts pre-processing of the received coding coefficient vectors and symbol vectors to maintain a specific "shifted version" of the reduced row echelon form of the receiver coding coefficient matrix. After the pre-processing, the symbols can be decoded with a standard block-based RLNC decoder.

Our extensive evaluations indicate that CRLNC achieves essentially the same levels of reliability (packet loss probability) as block based RLNC. In contrast to block based RLNC, which has relatively high in-order delivery delay, CRLNC achieves essentially the same low in-order delivery delays as sliding window RLNC. However, CRLNC does not have the prohibitive computational complexity and memory demands of sliding window approaches without feedback; instead, CRLNC has essentially the same low computational complexity and memory requirements as block based RLNC.

The remainder of this paper is structured as follows. Section II provides background and a literature review on related RLNC approaches. Section III gives an overview of the proposed CRLNC approach and describes the CRLNC encoding mechanism and packet format. Section IV introduces the CRLNC decoding mechanisms. Section V evaluates the performance of the proposed CRLNC in comparison to existing RLNC approaches. Section VI summarizes the conclusions from this study and outlines future work directions.

II. BACKGROUND AND RELATED WORK ON RLNC

A. OVERVIEW

RLNC [4], [32] linearly combines source (original data) symbols over a Galois field $GF(q)$ to create coded (redundant) symbols. Source symbols can be represented as row vectors of elements of $GF(q)$. In typical network settings, such as Ethernet, with a maximum transfer unit of 1500 bytes and a Galois field size $q = 2^8$, an Ethernet frame is represented using $m = 1500$ elements of $GF(2^8)$. Shorter frames are padded with zeros at the end.

We note that one research direction on network coding for low-delay communication has developed alternative coding approaches, e.g., coding based only on XOR operations, which essentially corresponds to networking coding for the special case of the binary Galois field $GF(2)$ [33]–[39]. In contrast, we consider RLNC with a general Galois field $GF(q)$, which achieves the optimal throughput [4], [35] and has negligible linear dependencies of the coding coefficient vectors for sufficiently large Galois fields, e.g., $GF(2^8)$. Another research direction has explored adaptations of various network coding parameters based on feedback from the receivers to the sender, see e.g., [40]–[45]; in contrast, we consider a network setting without feedback. A few related studies have examined network coding in specific networking contexts, e.g., in the context of delay-tolerant networks in [46], for video delivery in [47] and [48], for industrial networks in [49] and [50], and for sensor networks in [51]. We consider a general network context without feedback.

B. BLOCK BASED RLNC

Block based RLNC [24], [25] groups source symbols into equally sized "blocks" of g subsequent source symbols each. Such a block is also called a "generation", and g is called the generation size or block size. Conventional block based RLNC creates n_c, $n_c \geq 0$, coded symbols to protect the block of g source symbols. The coding step can be expressed as matrix multiplication of a coefficient matrix $\mathbf{C}$ (consisting of $g + n_c$ rows and g columns of coefficients in $GF(2^8)$) with the source symbol matrix $\mathbf{X}$ (consisting of g rows and m columns). For systematic block based RLNC [26]–[28], the top g rows of the coefficient matrix $\mathbf{C}$ consist of a $g \times g$ identity matrix, whereas the bottom n_c rows of $\mathbf{C}$ represent the random coefficients for generating the redundant coded symbols. The result of the matrix multiplication

$$\mathbf{Y} = \mathbf{C}\mathbf{X} \tag{1}$$

gives for systematic block based RLNC g rows containing the original source symbols, followed by n_c coded symbols, whereby each symbol consists of m elements (bytes with $GF(2^8)$) Note that for conventional block based RLNC, the coding window size is effectively $w_e = g$ and the code rate is $R = g/(g + n_c)$, as illustrated for $g = 4$ in Fig. 1(a).

A plethora of studies has examined the throughput-delay performance of block based RLNC, see e.g., [52]–[67], including for specific link layers, such as the LTE and WiMAX link layers [68]–[71]. Dynamic adaption of the generation size of block based RLNC has been examined in [23] and [72]–[74], while multi-generation mixing, which jointly encodes the source symbols from multiple successive generations, has been studied in [75]–[78] and RLNC with overlapping blocks has been investigated in [79] and [80]. The Pace variation [29] of block based RLNC uniformly distributes the n_c coded symbols among the g source symbols to enable fast recovery of dropped symbols without waiting for the end of the generation. When applying Pace to the example in Fig. 1(a), the first coded symbol is shifted up to linearly combine the first two source symbols in a generation, whereas the second coded symbol in a generation still linearly combines all $g = 4$ source symbols in a generation. Thus,

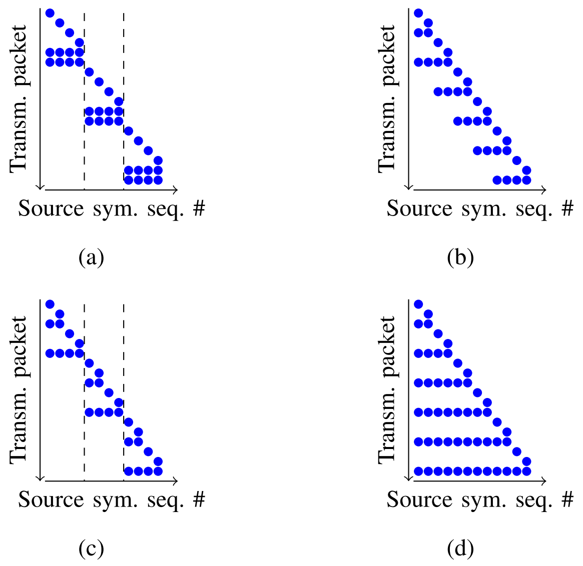


Fig. 1. Examples for the compared systematic RLNC coding schemes with the code rate $R = 2/3$. Systematic (uncoded) source symbols are represented by a single blue dot on a horizontal line. Multiple blue dots on a horizontal line indicate the specific source symbols that are combined to generate a coded symbol. E.g., block-based RLNC in part (a) combines the first four source symbols to generate each of the first two coded symbols. (a) Systematic block-based RLNC, $g = 4$. (b) Finite sliding window, CRLNC, $w_e = 4$, $n_s = 2$. (c) Pace RLNC, $g = 4$. (d) Infinite sliding window, $n_s = 2$.

the first two source symbols are protected by two coded symbols, whereas the last two source symbols are protected by only one coded symbol, as illustrated in Fig. 1(c). Overall, block based RLNC approaches are not well suited for packet streaming applications as they introduce either high delays or compromise the reliability.

C. SLIDING WINDOW RLNC

Sliding window RLNC [30], [81]–[84] does not group the source symbols into artificial blocks or generations. Instead, all source symbols within a consecutive sequence of source symbols are considered for creating coded symbols. This consecutive sequence of source symbols is referred to as *sliding window*. Appending new source symbols to the bottom (end) of this sequence of symbols is commonly referred to as *opening* the window. In contrast, removing source symbols from the top (beginning) of the sequence is referred to as *closing* the window. This sliding window RLNC has demonstrated superior delay properties for streaming applications compared to block based RLNC [30], [81]–[84]. However, the existing sliding window RLNC studies generally assume an infinite sliding window, or a sliding window of dynamic size, which is closed by feedback [85]–[91]. Reliable feedback is not available in some networks. Without feedback, dynamic window approaches need to store all symbols, i.e., require essentially an infinite sliding window. An infinite sliding window is impractical, mainly due to prohibitive storage requirements. Therefore, we propose CRLNC, a practical sliding window approach with a fixed-sized window that can be efficiently implemented without feedback. CRLNC combines the good delay properties of sliding window RLNC with the efficiency and reliability properties of conventional block based RLNC.

The code rate in sliding window RLNC is commonly controlled by the number of source symbols n_s that are sent before one coded symbol, i.e., the code rate is $R = n_s/(n_s + 1)$. More specifically, for systematic infinite sliding window RLNC, n_s uncoded source symbols are sent followed by one coded symbol, as illustrated for $n_s = 2$ in Fig. 1(d). A given coded symbol is obtained by linearly combining all preceding source symbols of the considered source symbol sequence.

To the best of our knowledge, finite sliding window RLNC has received relatively little research attention to date. An initial comparison of a finite sliding window RLNC approach with a block based forward error correction (FEC) approach based on Reed-Solomon codes has recently been presented in [92]. In contrast, we focus on RLNC throughout and consistently compare block based RLNC with finite and infinite sliding windows. We consider the in-order delay over burst error (two state) channels, whereas only a memoryless channel without the in-order requirement was considered in [92]. In addition, we provide detailed specifications of efficient practical algorithms for finite sliding window RLNC encoding and decoding.

III. PROPOSED CATERPILLAR RLNC (CRLNC): OVERVIEW AND ENCODING

For our proposed CRLNC, we use a *finite* sliding encoding window of size w_e, $w_e \geq n_s$. The CRLNC encoder uses the last w_e source symbols to create a coded symbol, while the code rate is fixed at a prescribed $R = n_s/(n_s + 1)$ as in conventional sliding window RLNC. A CRLNC example with encoding window size $w_e = 4$ and code rate $R = 2/3$, which implies $n_s = 2$ uncoded source symbols between two successive coded symbols, as illustrated in Fig. 1(b). We note that the code rate R accounts for the coding overhead due to one coded symbol per n_s source symbols. The code rate R does not account for the overhead required for the packet headers that are introduced in Section III-B and illustrated in Fig. 2.

A. SETTING AND OVERVIEW

We focus on network streams where a consecutive sequence of source symbols with source symbol sequence number i, $i = 0, 1, 2, \ldots$, are to be transmitted in order, from a single source to one or multiple destinations. We focus on intra-session RLNC which combines symbols from a single given stream to create redundant coded symbols. We propose a systematic sliding window RLNC scheme with a prescribed finite encoding window size w_e in terms of the number of successive source symbols that are combined to form a coded symbol. The main notations are summarized in Table 1. The coded symbols are created using RLNC [4], combining only the source symbols within the sliding encoding window of fixed size w_e. The window size constraint w_e limits the computational complexity, the required memory at all

header					payload
blk. #	c_i^η				
0	1	0	0	0	source symbol 0
0	0	1	0	0	source symbol 1
0	0	0	1	0	source symbol 2
0	0	0	0	1	source symbol 3
0	c_0^1	c_1^1	c_2^1	c_3^1	coded symbol 0–3
0	c_0^2	c_1^2	c_2^2	c_3^2	coded symbol 0–3
1	1	0	0	0	source symbol 4
1	0	1	0	0	source symbol 5
1	0	0	1	0	source symbol 6
1	0	0	0	1	source symbol 7
1	c_4^3	c_5^3	c_6^3	c_7^3	coded symbol 4–7
1	c_4^4	c_5^4	c_6^4	c_7^4	coded symbol 4–7

(a)

header					payload
seq. #	c_i^η				
0	1	0	0	0	source symbol 0
1	0	1	0	0	source symbol 1
1	c_0^1	c_1^1	0	0	coded symbol 0–1
2	0	0	1	0	source symbol 2
3	0	0	0	1	source symbol 3
3	c_0^2	c_1^2	c_2^2	c_3^2	coded symbol 0–3
4	1	0	0	0	source symbol 4
5	0	1	0	0	source symbol 5
5	c_4^3	c_5^3	c_2^3	c_3^3	coded symbol 2–5
6	0	0	1	0	source symbol 6
7	0	0	0	1	source symbol 7
7	c_4^4	c_5^4	c_6^4	c_7^4	coded symbol 4–7

(b)

Fig. 2. Packet format consisting of header (containing block number for block based RLNC and packet sequence number s_p for CRLNC and coding coefficients c_i^η) and payload (containing uncoded source symbol or coded symbol) for example parameters of Fig. 1. The coding coefficient c_i^η specifies how source symbol i is included in the linear combination to form the corresponding (ηth) coded symbol. The coefficient of the latest source symbol is highlighted through shading in each row. (a) Block RLNC packets for $g = 4$, $R = 2/3$. (b) Caterpillar RLNC (CRLNC) packets for $w_e = 4$, $n_s = 2$, $R = 2/3$.

TABLE 1. Summary of main notations.

	RLNC Parameters
g	Block size [in number of symbols] of conv. block based RLNC
n_s	Number of consecutive uncoded source symbols preceding a coded symbol in CRLNC
w_e	Encoding window size [in number of symbols] of CRLNC
w_d	Decoding window size [in number of symbols] of CRLNC
R	Code rate; $R = n_s/(n_s + 1)$ for CRLNC
	Packet Sequence Numbers
i	Source symbol sequence number at sender; $i = 0, 1, 2, \ldots$
s_p	Packet sequence number of transmitted packet; $s_p = i$ for uncoded source packet; $s_p =$ highest transmitted i for coded packet
s_d	Decoder sequence number; $s_d =$ highest received s_p

network nodes, and the overhead in each packet. We proceed to introduce the CRLNC encoding and the format of the coded packets in Section III-B. The CRLNC decoder is introduced in Section IV.

B. CRLNC ENCODING AND PACKET FORMAT

A packet carries a symbol, i.e., either an uncoded source symbol or a coded symbol, over the network channel. A packet consists of a header, which contains auxiliary information and the coding coefficients, as well as the packet body (packet payload), which contains the symbol data. We note that there are various sophisticated header compression schemes, e.g., [81], [93], [94] However, for simplicity, we consider only a simple elementary header representation in this study. For ease of terminology, we refer to a packet carrying an uncoded source symbol as a ‘‘source packet’’ and to a packet carrying a coded symbol as a ‘‘coded packet’’.

For block based RLNC, the auxiliary information is the block number as indicated in the left-most column in the example illustrated in Fig. 2(a). As illustrated in Fig. 2(a) for $g = 4$, the coding coefficient row vector has the fixed size of g coefficients. For an uncoded (systematic) source symbol, the coding coefficient vector is a unit vector with a one at the respective index representing the position of the source symbol within the block, e.g., for source symbol $i = 2$ in Fig. 2(a), the coefficient vector is $(0, 0, 1, 0)$. For a coded packet, the coding coefficient c_i^η indicates how source symbol i is included in the linear combination of the source symbols in the block to obtain the ηth coded packet. For block based RLNC, the coefficient vector in a coded packet can be represented as a simple ordered sequence of g coefficients c_i^η, e.g., as c_4^4, c_5^4, c_6^4, c_7^4, in the bottom packet in Fig. 2(a), since the correspondence of a coding coefficients to the correct respective source symbol is signalled through the block id number and the ordering of the c_i^η in the packet header.

In contrast, as a sliding window approach, CRLNC lacks the block id number. We propose to signal the correspondence between coding coefficients and source symbols as follows. First, we define the *packet sequence number* s_p: For a source symbol, which is transmitted uncoded with the considered systematic RLNC, the packet sequence number s_p is set equal to the source symbol sequence number i. For a coded packet, the packet sequence number s_p is set to the highest source symbol sequence number i that has been considered for forming the coded symbol. In other words, from the perspective of a coded packet, the packet sequence number s_p indicates the last source symbol that was included in the linear combination to form the coded symbol carried in the considered coded packet.

Second, we place the coding coefficient c_i^η corresponding to source symbol i at position $i \mod w_e$ among the w_e coding coefficient row vector positions, i.e., the positions $0, 1, \ldots, w_e - 1$ for coding coefficients in the packet header, as illustrated for $w_e = 4$ in Fig. 2(b). For instance, the coding coefficients corresponding to source symbol $i = 5$ are placed at position $5 \mod 4 = 1$, i.e., at the second position from the left among the coding coefficient positions in the packet header. With this format, the coding coefficient c_i^η for a specific source symbol number i and coded packet number η can always be found at a well-defined prescribed position. This specific positioning of the c_i^η will be utilized by the decoder to efficiently build the decoding matrix.

We note that both schemes exhibit an overhead of w_e elements of $GF(q)$ for the coding coefficients, which could be compressed [81], [93], [94]. In addition, the packet header has to carry the block number for block RLNC, while CRLNC requires the packet sequence number s_p in the packet header. With a block containing g source symbols, the packet sequence number s_p field in the packet header requires $\log_2 g$ more bits than the block number field to index the same total number of source symbols. This additional CRLNC overhead of $\log_2 g$ bits is negligible for typical communication settings.

IV. CRLNC DECODING

A. GENERAL CONVENTIONAL DECODING APPROACH

In general, the decoding step is the inversion of the encoding. For a block based scheme, the receiver collects g linearly independent coding coefficient vectors in the receiver coding coefficient matrix **R** with the corresponding matrix of received symbols $\bar{\mathbf{Y}}$. The source symbols can be reconstructed by Gaussian elimination that computes the coefficient matrix inversion and multiplication:

$$\mathbf{X} = \mathbf{R}^{-1}\bar{\mathbf{Y}} \tag{2}$$

with computational complexity $O(g^3)$ [95]. Note that this general decoding approach operates on one block at a time in constant memory, similar to the block based RLNC encoder. After a block is decoded, or a symbol from the next block is received, the matrices **R** and $\bar{\mathbf{Y}}$ can be cleared and reused for decoding the next block.

B. OVERVIEW OF CRLNC DECODING: ITERATIVE GAUSSIAN ELIMINATION

In contrast to the conventional general decoding of a block of g symbols, we iteratively update the receiver coding coefficient matrix and decode symbol by symbol. Formally, we let w_d, $w_d \geq w_e$, denote the decoding window size in number of considered successive packets. For ease of explanation, we initially focus on the special case where the decoding window equals the encoding window, i.e., $w_d = w_e$; we subsequently explain the extension of the presented decoding approach to general w_d values in Section IV-D. We let **R** denote the receiver coding coefficient matrix consisting of w_d rows and w_d columns of $GF(q)$ elements.

Our overall strategy is to keep the receiver coding coefficient matrix **R** in a "shifted version" of the conventional reduced row echelon form. The conventional reduced row echelon form (row canonical form) [95] has (*i*) nonzero rows with at least one non-zero element above the rows with all zeros, and (*ii*) the leading coefficient, i.e., the first nonzero element from the left (often referred to as the pivot), of a nonzero row strictly to the right of the pivot of the row above the considered row, and (*iii*) each pivot equal to one as the only nonzero element in its column. The "shifting" moves the position of the pivot of the top-most row of the matrix **R** relative to the fixed memory locations holding the matrix elements as the sequence number of the received packets increases, as explained in detail in Section IV-C. The shifting is neglected for now, as we explain the overall CRLNC decoding strategy.

The CRLNC decoding strategy is to iteratively solve the system of linear equations of the form of Eqn. (2) with Gaussian elimination. More specifically, let $\mathbf{R}_t$ denote the receiver coding coefficient matrix in reduced row echelon form at a given time t. Suppose that a new packet is received over the communication channel at time $t + 1$. The updated receiver coding coefficient matrix $\mathbf{R}_{t+1}$ is formed as follows. First, the newly received coding coefficient (row) vector is inserted into the matrix $\mathbf{R}_t$ (the specific inserting procedure will be explained in Section IV-C), resulting in an intermediate matrix $\mathbf{R}'_t$ (that is no longer in reduced row echelon form). Then, Gaussian elimination is applied to transform the intermediate matrix $\mathbf{R}'_t$ into reduced row echelon form to yield $\mathbf{R}_{t+1}$. This Gaussian elimination step can be conducted with a slightly modified version of the standard block-based RLNC tools [96]; specifically, the standard RLNC tools expect the top-most pivot in the upper left of the coding coefficient matrix, whereas our modification allows for arbitrary column positions of the top-most pivot.

If a row of the obtained receiver coding coefficient matrix $\mathbf{R}_{t+1}$ consists of a single one and all remaining row elements are zero, then the corresponding symbol has been successfully decoded. The successfully decoded symbol can then be delivered to the next higher communication protocol layer, possibly after ensuring that in-order delivery constraints are satisfied.

In conventional infinite sliding window RLNC, the size of the receiver coding coefficient matrix is not bounded. Every preceding received packet in a packet sequence could be required for decoding a given packet. In contrast, we design CRLNC decoding to operate with a receiver coding coefficient matrix **R** with limited size of w_d, $w_d \geq w_e$, rows and w_d columns of $GF(q)$ elements (and a corresponding received symbol matrix $\bar{\mathbf{Y}}$ holding at most w_d rows and m columns of $GF(q)$ elements). In order maintain the size limit of the receiver coding coefficient matrix **R**, we remove a coding coefficient row vector from **R** if the coding coefficient row vector can no longer be fully held in **R** as the sequence number of received packets advances. In particular, when the decoder receives a new packet with sequence number s_p, all rows with a nonzero coefficient at a column position corresponding to a sequence number less than $s_p - w_d + 1$ are removed from **R**, as explained in detail in Section IV-C.2.

C. INSERTING ARRIVING PACKETS INTO R: SHIFTING OF THE ROW ECHELON FORM

This section explains the details of inserting the coding coefficient row vector from packets arriving to the receiver into the receiver coding coefficient matrix **R**. The symbols from arriving packets are correspondingly inserted into the receiver symbol matrix $\bar{\mathbf{Y}}$. More specifically, we introduce the preprocessing manipulations that have to be made to the receiver coding coefficient matrix **R** such that the infinite stream of incoming packets with the proposed CRLNC approach of finite sliding window RLNC can be processed with the modified block based RLNC decoder outlined in Section IV-B. For ease of explanation, we initially consider a decoding window equal to the encoding window, i.e., $w_d = w_e$. Recall from Section III-B that s_p denotes the sequence number of an arriving packet. We define the sequence number s_d of the decoder as the highest packet sequence number that the decoder has received thus far. When a new packet arrives, the matrix **R** has to be manipulated based on the sequence number of the packet s_p and the sequence number of the decoder s_d. There are three distinct cases to consider.

1) SAME SEQUENCE NUMBER: $s_p = s_d$

An arriving packet with a sequence number s_p equal to the decoder sequence number s_d can, for instance, be a coded packet that follows an uncoded packet, e.g., the coded packet

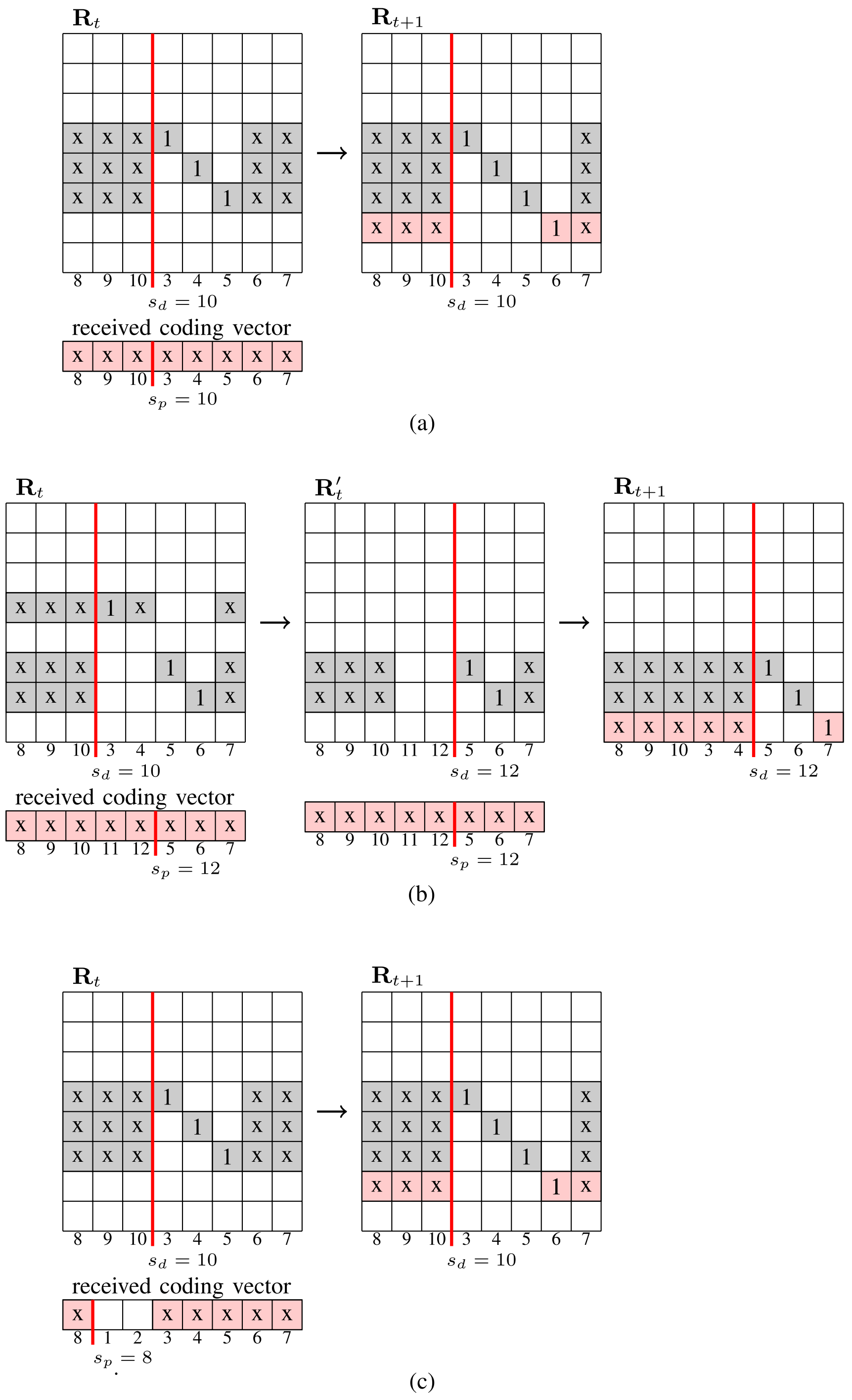


Fig. 3. Example illustrations of the three different cases for inserting the coding coefficient row vector from a newly received packet with sequence number s_p into the receiver coding coefficient matrix $\mathbf{R}_t$ with decoder sequence number s_d. In the example, the encoding window size w_e and decoding window size w_d are equal to eight, i.e., $w_e = w_d = 8$. The red line represents the latest sequence number. White spaces indicate zero-valued coefficients. (a) Same sequence number $s_p = s_d$: packet is inserted. (b) $s_p > s_d$: increment s_d, erase rows with nonzero coefficients at sequence numbers less than $s_p - w_d + 1 = 12 - 8 + 1 = 5$. (c) $s_p < s_d$: packet can only be inserted because of the zero coefficients at sequence number positions 1 and 2.

carrying the coded symbol 0–3 in Fig. 2b. The coding coefficient row vector of the received packet aligns perfectly with the decoder coding coefficient matrix $\mathbf{R}_t$, that is, the column positions in both the packet and the decoder matrix correspond to the same sequence numbers, as illustrated for an example with $w_d = 8$ for $s_p = s_d = 10$ in Fig. 3a.

Thus, the packet can be directly inserted into the present form of the receiver coding coefficient matrix $\mathbf{R}_t$ to obtain the intermediate matrix $\mathbf{R}'_t$. The decoder then applies Gaussian elimination to $\mathbf{R}'_t$ so as to obtain the updated matrix $\mathbf{R}_{t+1}$ in reduced row echelon form.

2) NEW PACKET: $s_p > s_d$

If the newly arriving packet has a higher sequence number s_p than the decoder sequence number s_d, then the decoder sequence number needs to be advanced, implying a shift of the reduced row echelon form of $\mathbf{R}_t$, as illustrated in Fig. 3b for $s_p = 12$ and $s_d = 10$. Moreover, coding coefficient row vectors that cannot be fully held in $\mathbf{R}_{t+1}$ after the shift are deleted. That is, coding coefficient rows with non-zero elements at column positions corresponding to sequence numbers less than $s_p - w_d + 1$ need to be removed. In particular, in the example in Fig. 3b, the first (top-most) coding coefficient row in $\mathbf{R}_t$ on the left, has a one at the column position corresponding to sequence number 3, which is less than $s_p - w_d + 1 = 5$. Hence, this top-most coding coefficient row can no longer be fully held and needs to be removed as the decoder sequence number is advanced to $s_d = 12$.

The remaining coding coefficient vectors in $\mathbf{R}_t$ that can still be fully held after the advance of the decoder sequence number s_d remain at their memory locations. The decoder sequence number s_d is advanced to s_p, resulting in the shifted reduced row echelon form illustrated in the middle in Fig. 3b. Now, the column positions of decoder matrix and received packet align to correspond to the same sequence numbers and the packet can be inserted into the decoder matrix to obtain the intermediate matrix $\mathbf{R}'_t$.

Subsequently, Gaussian elimination is applied to $\mathbf{R}'_t$ so as to restore the reduced row echelon form in $\mathbf{R}_{t+1}$ on the right in Fig. 3c. Notice that with the advance of the decoder sequence number to $s_d = 12$, the reduced row echelon form has shifted to have the top-most pivot in the position corresponding to sequence number 5. Also, the receiver coding coefficient matrix now (after the advance to $s_d = 12$) covers the sequence number range 5 through 12.

3) OLD PACKET: $s_p < s_d$

A newly arriving old packet with $s_p < s_d$ can only be used if all non-zero coding coefficients of the coding coefficient row vector (covering the sequence number range $s_p, s_p - 1, \ldots, s_p - w_e + 1$) of the arriving packet fall within the sequence number range $s_d, s_d - 1, \ldots, s_d - w_d + 1$ covered by the decoder matrix. In other words, the old packet can only be used if all coefficients outside of the sequence number range that is currently covered by the decoder window are zero. When checking these sequence numbers it is critical to keep in mind the specific placement of coding coefficients according to the CRLNC packet format introduced in Section III-B, i.e., the coefficient c_i^η corresponding to source symbol sequence number i is placed in column position i mod w_e.

In the example with $s_p = 8$ and $s_d = 10$ illustrated in Fig. 3c, the newly arriving packet has zeros in the column positions corresponding to sequence numbers 9 and 10 in $\mathbf{R}_t$ (these column positions correspond to sequence numbers 1 and 2 from the packet's perspective). Thus, the newly arriving old packet can be inserted into $\mathbf{R}_t$ without losing any non-zero coding coefficient. The situation would be different if there were non-zero elements in these column positions corresponding to sequence numbers 1 and 2 from the packet's perspective. Then, the packet could not be inserted into $\mathbf{R}_t$ since the non-zero elements in the column positions corresponding to sequence numbers 1 and 2 from the packet's perspective would align with the sequence numbers 9 and 10 from the decoder's perspective.

D. EXTENSION TO DECODING WINDOW $w_d > w_e$

The CRLNC decoding approach has so far been described for the special case when the decoding window size w_d equals the encoding window size w_e. The CRLNC decoding operates analogously for decoding windows larger than the encoding window, as illustrated in Fig. 4, for an example with $w_d = 8$ and $w_e = 4$. The longer decoding window can permit the decoding of more packets and reduce the loss probability, as quantitatively examined in Section V-D. In particular, newly received packets may help in the successful decoding of older packets that are still kept in the decoding window. In the example illustrated in Fig. 4, the decoding of the rows with the ones in the column positions corresponding to sequence numbers 5 and 6 is aided by the newly received packet with sequence number $s_p = 10$, since the second non-zero elements in these rows at sequence number 7 receive an additional non-zero element at sequence number 7 in the newly received packet. (Note that with a decoding window of $w_d = w_e = 4$, the rows with the ones at sequence numbers 5 and 6 would already have been erased—and consequently constitute losses—since for $s_d = 10$, the decoding window would only cover the sequence numbers 10, 9, 8, and 7.)

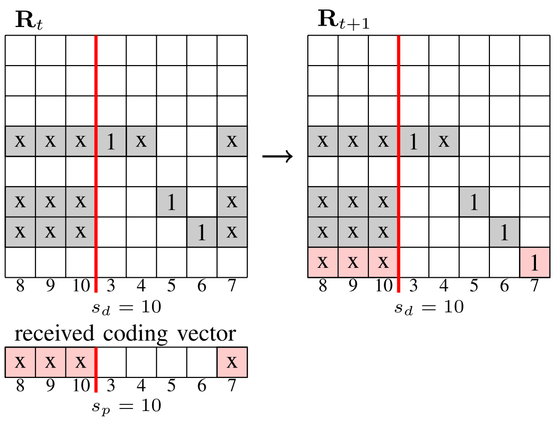


Fig. 4. Example illustration of decoding with $w_d = 8 > w_e = 4$.

However, when the sequence number range covered by the encoding windows of the newly received packets has moved past the second nonzero coefficient of an older packet, then the decoding of the older packet is not further aided by the newly received packets. Consider for instance, the decoding of the top-most row in Fig. 4 with a one in the column position corresponding to sequence number 3 and the second nonzero coefficient at sequence number 4 (this scenario can arise when the packet with sequence number 4 is dropped by the channel). The decoding of this top-most row is not aided by the newly received packet with sequence number $s_p = 10$ as the newly received packet only covers the sequence numbers from 7 to 10.

E. SEQUENCE NUMBER REPRESENTATION

The sequence numbers need to uniquely identify the packets within the decoding window w_d, $w_d \geq w_e$. The sequence number behaviors of sliding window protocols have been extensively studied for communication networks, see e.g., [97]–[100]. For the general setting of a transport channel that can lose, duplicate, and reorder sent packets, the sequence number space, i.e., the number S of unique sequence numbers, needs to cover at least twice the window size plus the maximum lifetime of packet in the network divided by the minimum packet transmission time [99]. We envision CRLNC RLNC mainly for low-latency applications, which will typically have limited packet lifetimes in the network. Nevertheless, as for the wide range of existing network protocols based on the sliding window protocol principle, the sequence number space requires careful consideration in our proposed CRLNC.

F. CRLNC DECODING COMPLEXITY ANALYSIS

By design, the CRLNC decoder has a storage complexity of w_d^2 elements of $GF(2^8)$ for the receiver coding coefficient matrix $\mathbf{R}$ and mw_d elements of $GF(2^8)$ for the corresponding symbols (packet data) in $\bar{\mathbf{Y}}$. In addition, the decoder has to store the decoder sequence number s_d, which requires $\log_2 S$ bits for the sequence number range S. Note that by the design of the decoding process, the receiver symbol matrix $\bar{\mathbf{Y}}$ holds the decoded symbols until they can be delivered in order to the receiving application (or a symbol loss is determined).

In terms of computational complexity, the processing of g arriving packets at the decoder requires g times the execution of the coding coefficient row vector insertion described in Section IV-C. Each coding coefficient row vector insertion requires a constant computational complexity for advancing the decoder sequence number and deleting rows, see Section IV-C.2. In particular, for each increment in the received packet sequence number s_p, at most one row will be removed from the matrix $\mathbf{R}_t$. This is due to the row echelon form of the matrix $\mathbf{R}_t$. More specifically, by the definition of row echelon form, a matrix $\mathbf{R} \in GF(q)^{m\times n}$ with columns $c_1, c_2, \ldots, c_n$ in row echelon form has at most x rows with nonzero elements among the columns $c_1, c_2, \ldots, c_x$.

In addition, each coding coefficient row vector insertion requires a computational complexity of $O(w_d^2)$ for re-creating the reduced row echelon form, i.e., for completing the triangulation steps followed by the backward substitution steps to re-create the reduced row echelon form. Thus, the overall computational complexity for processing a sequence of w_d source packets in the CRLNC decoder is $O(w_d^3)$, which is equivalent to the computational complexity of a conventional block based RLNC decoder. We further quantify the computational complexity by evaluating the number of symbol vector operations for both CRLNC and block based RLNC decoding in Section V-E.

V. CRLNC EVALUATION

A. SIMULATION SETUP

For the CRLNC evaluation, we implemented a time-slotted, stochastic discrete event simulator. The simulator models a sender-receiver process with a binary erasure channel. We compare the packet loss probabilities and the in-order packet delays of the proposed finite sliding window CRLNC approach with conventional block (generation) based RLNC, with the Pace approach [29], and with infinite sliding window RLNC [30], [81]. We define a packet loss to occur when a source symbol (source packet) has not been decoded when it is shifted out of the decoding window (in CRLNC), or when the next block is started (in block based RLNC), or when the simulation replication is completed (in infinite window RLNC). That is, a packet loss occurs when a sent source symbol cannot be recovered (decoded) by the decoder at the receiver. The packet loss probability is defined as the long-run proportion of the number of source symbols that have not been decoded to the total number of sent source symbols.

We consider in-order packet delivery at the receiver. The decoder delivers a newly decoded packet to the next higher protocol layer if the immediately preceding packet has already been delivered or the decoder has determined that the preceding packet cannot be recovered anymore. We define the in-order packet delay as the number of elapsed time slots from the time instant when a source symbol is pulled by the RLNC encoder (at the sender) from the next higher protocol layer (e.g., the network layer) to the time instant when the source symbol is delivered in order by the decoder to the next higher protocol layer at the receiver. We assume that the RLNC encoding and decoding computation times [101], [102] and the channel propagation delay are negligible compared to a packet transmission time slot.

In the simulator, we considered an infinite field size, which avoids linear dependencies of the coefficient vectors. That is, every received coded packet increases the rank of an incomplete decoding matrix. In practice, a sufficiently large finite field [30], e.g., $GF(2^8)$, can be used to achieve equivalent results, as we have verified through comparisons between the simulator and a proof-of-concept implementation based on Kodo [96] with $GF(2^8)$.

We consider a discrete-time Gilbert-Elliot channel model [103], [104] with γ denoting the transition probability from the "good" state, where all packets arrive, to the "bad"

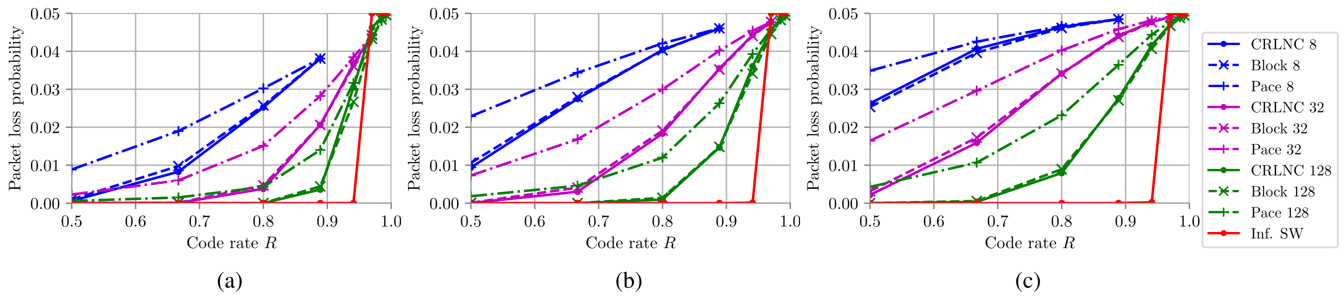


Fig. 5. Packet loss probability as a function of code rate R for Caterpillar RLNC (CRLNC) with encoding window sizes w_e, $w_e = \{8, 32, 128\}$ compared to conventional block (generation) based RLNC and Pace with block sizes $g = w_e$ and infinite sliding window RLNC for different expected numbers $E[B]$ of successive channel packet drops. Fixed parameters: CRLNC decoding window $w_d = w_e$; channel packet drop probability $\pi_B = 5$ %. (a) $E[B] = 2$. (b) $E[B] = 4$. (c) $E[B] = 8$.

state, where all packets are dropped. The parameter β denotes the transition probability from the "bad" state to the "good" state. We set the transition probabilities γ and β to give a prescribed steady-state probability for the "bad" state $\pi_B = \gamma/(\gamma + \beta)$, which corresponds to the expected long-run channel packet drop probability, and the expected sojourn time in the bad state $E[B] = 1/\beta$, which corresponds to the expected number of successive packet drops on the channel.

For each evaluation scenario, we conduct 10^3 independent replications, each simulating the transmission of 10^6 source symbols (source packets). The resulting 95 % confidence intervals are smaller than 2 % of the corresponding sample means and are not shown in the plots to avoid visual clutter.

B. PACKET LOSS

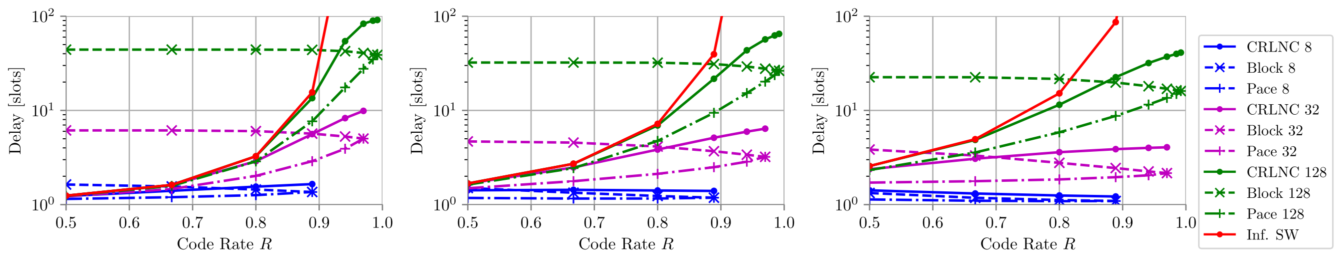
Fig. 5 shows the packet loss probabilities as a function of the code rate R for different expected numbers $E[B]$ of successive packet drops on the channel. We observe from Fig. 5 that CRLNC with a given encoding window size w_e achieves (*i*) slightly lower loss probabilities than conventional block (generation) based RLNC with block size $g = w_e$, and (*ii*) substantially lower loss probabilities than Pace with block size $g = w_e$. Block based RLNC is restricted to attempt the recovery of packets dropped on the channel within each given individual block. In particular, if the number of received coded packets is equal to or larger than the number of dropped source packets within a block, then all dropped source packets can be recovered. On the other hand, if the number of dropped source packets within a block exceeds the number of received coded packets, then none of the dropped source packets can be recovered and become lost packets for the evaluation. Block based RLNC attempts the recovery on a block-by-block basis, i.e., moves the decoding window effectively in steps of complete blocks.

In contrast, CRLNC slides (advances) the decoding window as specified in Section IV-C; i.e., the decoding window advances by one sequence number position with a newly received in-order source packet, or by n sequence number positions if the preceding $n - 1$ source packets have been dropped. For each given position of the decoding window, CRLNC recovers dropped source packets if the number of received coded packets is equal to or larger than the number of dropped source packets within the particular range of packets covered by the decoding window. For small expected numbers $E[B]$ of successive channel packet drops, CRLNC thus advances the decoding window in smaller steps and has more chances for packet recovery than block based RLNC, which effectively advances the decoding window in steps of complete blocks, i.e., in steps of g source packets. Thus, CRLNC achieves slightly lower packet loss probabilities than block based RLNC.

The substantially higher Pace loss probability is due to the distribution of the coded packets among the source packets in Pace. This distribution of the coded packets provides only weak protection for the source packets at the end of a generation. Specifically, the first source packet is protected by all n_c coded packets of the generation, whereas the last source packet is protected by only one coded packet. This distribution of the coded packets results in non-uniform protection levels for the source packets in a generation. That is, source packet towards the end of a generation experience a higher loss probability than source packets near the beginning of a generation [29]. Thus, the source packets in Pace are on average protected by fewer coded packets than in conventional block based RLNC and in CRLNC. Pace leads therefore to substantially higher packet loss probabilities, as observed in Fig. 5.

Fig. 5 also confirms that for increasing encoding window size w_e and block size g, the CRLNC and block based RLNC, respectively, approach the low loss probability achieved by infinite sliding window RLNC. Infinite sliding window RLNC, in turn, exhibits a jump to high packet loss probabilities when the code rate R becomes so high that the long-run ratio of the number of coded packets to the number of source packets drops below the channel packet drop probability π_B. In other words, when the code rate R (ratio of source symbols to sum of source symbols and coded packets) exceeds the theoretical channel capacity of $1 - \pi_B$ packets per time slot then there are too few coded packets to recover the source packets that are dropped by the channel.

We furthermore observe from Fig. 5 that CRLNC and block based RLNC exhibit higher packet loss probabilities



(a) (b) (c)

Fig. 6. Mean in-order packet delay as a function of code rate R for range of CRLNC encoding window sizes w_e compared to conventional block (generation) based RLNC and Pace with block sizes $g = w_e$ and infinite sliding window RLNC for different expected numbers $E[B]$ of successive channel packet drops. Fixed parameters: CRLNC decoding window $w_d = w_e$; channel packet drop probability $\pi_B = 5$ %. (a) $E[B] = 2$. (b) $E[B] = 4$. (c) $E[B] = 8$.

for longer expected bad channel sojourn times $E[B]$, i.e., for higher mean numbers of successive packet drops on the channel. Infinite sliding window RLNC is not affected by $E[B]$ since any dropped source symbol can be recovered over a long time horizon of future coded packets (provided the code rate R is less than the channel capacity $1 - \pi_B$). In contrast, CRLNC and block based RLNC can only recover dropped source packets within the finite decoding window w_d. For low code rates R, $R < 1 - \pi_B$, CRLNC and block based RLNC can recover individual channel packet drops (occurring with probability π_B) that are sufficiently far spaced apart to have more coded packets than dropped source packets within the decoding window. However, increased expected bad channel sojourn times $E[B]$ tend to increasingly "clump" the channel packet drops together. Consequently, it becomes increasingly likely that there are more dropped packets than coded packets within the decoding window, making the recovery of channel packet drops impossible and leading to actual packet losses.

C. IN-ORDER PACKET DELAY

Fig. 6 plots the mean in-order packet delay as a function of the code rate R for different expected bad channel sojourn times $E[B]$. We observe from Fig. 6 that the infinite sliding window RLNC delay increases with the code rate R. With a higher code rate R, $R \rightarrow 1$, there are more $(R/(1-R))$ source packets between successive individual coded packets. Thus, the recovery of a dropped source packet has to wait longer until sufficiently many coded packets have been received. This waiting is exacerbated for long $E[B]$ when the channel typically drops multiple successive source packets.

For CRLNC, we observe from Fig. 6 for low code rates R the same low delays as with infinite sliding window RLNC, while achieving shorter delays than block based RLNC (except for the $E[B] = 8$ scenario). For increasing code rates R, the CRLNC delays flatten out while climbing slightly above the block based RLNC delays. For low code rates R, individual coded packets are interspersed frequently among the source symbols, e.g., with $R = 1/2$, every second packet is a coded packet, while for $R = 2/3$ every third packet is a coded packet. Thus, with infinite sliding window RLNC and CRLNC there are only short wait times for recovering source packets dropped by the channel with the next coded packets. In contrast, in block based RLNC all coded packets appear at the tail end of the block. Thus, any packet recovery has to wait until the end of the block. For instance, if the first source packet in a block of $g = 16$ source packets is dropped by the channel, then the source packet recovery has to wait until all subsequent 15 source packets and then enough coded packets (as many as source packets dropped by the channel) are received. Thus, larger generation sizes g lead to longer delays in block based RLNC, as observed from Fig. 6.

For high code rates R, packet losses become more likely. CRLNC determines a packet loss when the affected packet sequence number slides out of the decoding window, i.e., CRLNC can only detect a loss after advancing the decoding window by the full decoding window duration w_d. Block based RLNC determines a packet loss when there are not enough coded packets at the end of a block to recover the dropped source packets of the block. Thus, the wait time until the loss determination is upper bounded by the total number of packets in the block, but is shorter when the affected source packet is positioned later in the block. Overall, CRLNC incurs therefore slightly longer delays than conventional block based RLNC for high loss scenarios, which are typically not practically relevant.

Pace achieves somewhat shorter delays than CRLNC, whereby the Pace delays converge to the delays of conventional block RLNC for high code rates R. The shorter Pace delays compared to the CRLNC delays are due to a combination of the fast recovery of some dropped packets by the distributed coded packets in Pace and the faster loss determination in blocked based RLNC.

In order to obtain additional deeper insights into the delay dynamics for practically relevant loss scenarios, we plot in Fig. 7 the cumulative distribution function (CDF) of the packet delays. For Fig. 7, we set the code rate R to the highest value that achieves a packet loss probability below 0.1 % for CRLNC with $w_e = 64$. We observe from Fig. 7 that CRLNC achieves generally the same low delays as infinite sliding window RLNC up to a maximum delay value that corresponds to the determination of a packet loss (when the decoding window has slid beyond the affected sequence number in CRLNC, or when all coded packets at end of the block have been received in block based RLNC). This maximum delay

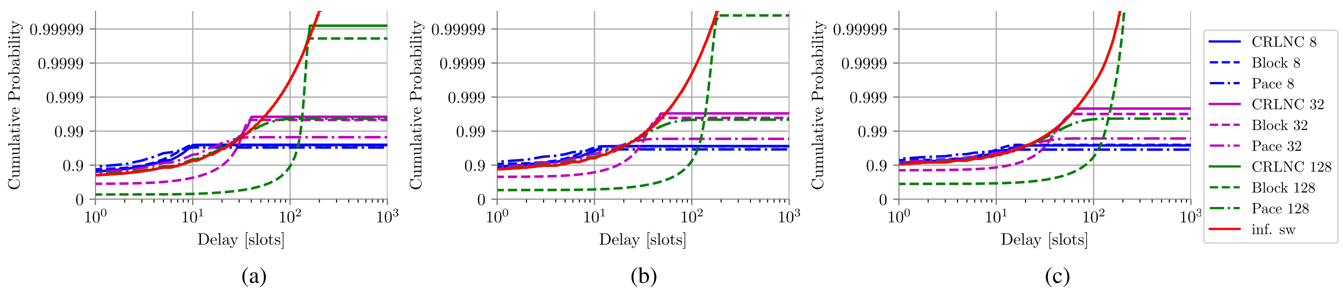


Fig. 7. **Cumulative distribution functions (CDFs) of the in-order packet delays for code rates R ensuring low losses for range of encoding window sizes w_e and block sizes g. Fixed parameters: CRLNC decoding window $w_d = w_e$; channel packet drop probability $\pi_B = 5$ %. The solid CRLNC curves essentially follow the infinite sliding window curve up to a maximum delay value that depends on the window size. (a) $E[B] = 2$, code rate $R = 0.8$. (b) $E[B] = 4$, code rate $R = 0.667$. (c) $E[B] = 8$, code rate $R = 0.5$.**

TABLE 2. **Effects of increasing the decoding window size w_d relative to the encoding window size w_e in CRLNC: Packet loss probability and delay for different encoding window sizes w_e. Fixed parameters: channel packet drop probability $\pi_B = 5$ %, $E[B] = 4$ successive channel packet drops on average, code rate $R = 2/3$.**

		w_d/w_e						
w_e		1	1.25	1.5	1.75	2	3	4
8	Loss [%]	2.75	2.46	2.41	2.40	2.40	2.40	2.40
	Delay [slots]	1.44	1.56	1.69	1.84	2.01	2.9	4.15
32	Loss [%]	0.30	0.15	0.10	0.086	0.082	0.080	0.080
	Delay [slots]	2.48	2.58	2.64	2.68	2.71	2.88	3.12
128	Loss [%]	$1.7 \cdot 10^{-3}$	$4.2 \cdot 10^{-4}$	$4.1 \cdot 10^{-4}$	$4.2 \cdot 10^{-4}$	$4.2 \cdot 10^{-4}$	$4.2 \cdot 10^{-4}$	$4.2 \cdot 10^{-4}$
	Delay [slots]	2.71	2.71	2.71	2.71	2.71	2.71	2.71

value grows with the decoding window size w_d, or block size g, and the average bad channel state sojourn time $E[B]$. Rare instances of maximum delays longer than w_d, or g, time slots arise when more than w_d, or g, successive packets are dropped by the channel. Such a rare scenario prevents the CRLNC decoder from advancing the sliding window until a new packet (with a sequence number more than w_d positions ahead) arrives; while the block based decoder waits until a new packet (with a higher block id number) arrives.

Importantly, we observe from Fig. 7 that for CRLNC, the probability for delivering source packets within a prescribed practical delay tolerance threshold is independent of the window size w_e. In contrast, for conventional block based RLNC, the probability for timely packet delivery drops as the block size g is increased. For instance, we observe from Fig. 7(b) that 97 % of the packets are delivered within 10 slots for $g = 8$. For $g = 32$, only 84 % of the packets are delivered within 10 slots, while only about half of the packets are delivered within 10 slots for $g = 128$. CRLNC consistently delivers about 96 % of the packets within 10 slots, independent of the window size. Thus, CRLNC allows for increasing the reliability, i.e., reducing the packet loss probability, through increasing the window size w_e without reducing the proportion of packets that are delivered within a prescribed delay tolerance threshold.

D. DECODING WINDOW w_d LARGER THAN ENCODING WINDOW w_e

Table 2 reports the packet loss probabilities and delays as a function of the ratio w_d/w_e of the decoding window size w_d to the encoding window size w_e. We observe from Table 2 that slight increases of the decoding window size w_d relative to the encoding window size w_e substantially reduce the loss probability, while incurring only modest delay increases. For instance, for $w_e = 32$, increasing w_d from 32 to 48 reduces the loss probability from 0.30 % to 0.10 %, while increasing the delay only from 2.48 slots to 2.64 slots. Further w_d increases beyond 1.5 w_e generally give only minute loss probability reductions while further increasing the delay. The delay increases are particularly pronounced for small encoding windows w_e which give relatively high loss probabilities. For instance, for $w_e = 8$, we observe a mean delay increase to 4.15 slots for $w_d = 4\ w_e$; this high delay is mainly due to the long delay for determining a loss in CRLNC. In contrast, for $w_e = 128$, the loss probabilities are very low, avoiding delay increases due to the determination of loss events. Overall, we conclude that a decoding window size w_d on the order of 1.5 times the encoding window size w_e achieves a good compromise between reducing the loss probability while maintaining short delays.

E. COMPUTATIONAL COMPLEXITY

To validate the practicality of our CRLNC approach, we evaluated the computational effort required for decoding. The matrix inversion and multiplication make RLNC decoding much more computationally demanding than RLNC encoding [101], [102]. Therefore, studies on RLNC computation typically focus on the decoding operation.

For conventional systematic block based RLNC, we used the implementation provided by the Kodo library [96],

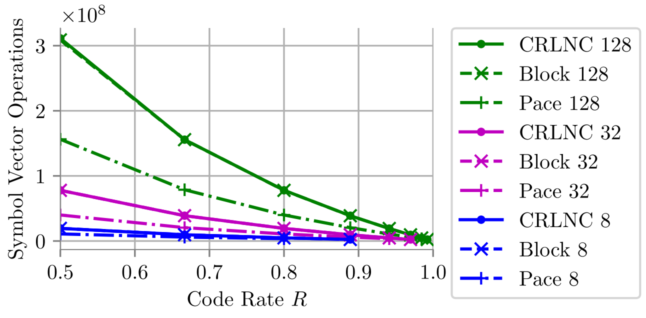


Fig. 8. Computational complexity: Average number of symbol vector operations for decoding operations in one simulation replication with 10^6 transmitted source packets as a function of code rate R for range of encoding window sizes w_e and block sizes g. Fixed parameters: CRLNC decoding window $w_d = w_e$; Channel packet drop probability $\pi_B = 5$ %; Mean number of $E[B] = 4$ successive channel packet drops.

which we also used as a basis for the Pace implementation. We implemented CRLNC on top of Kodo with the modifications described in Section IV-C. We did not evaluate the computational complexity of the infinite sliding window due to its prohibitive computational complexity when used without feedback for closing the decoding window. Following the computational complexity evaluation approach in [29] and [105], we count the number of operations for the decoding operations. Specifically, we count the number of symbol vector operations in one simulation replication of 10^6 transmitted source symbols and report averages over 10^3 independent replications. The resulting 95 % confidence intervals are less than 2 % of the sample means and are omitted from the plot. We observe from Fig. 8 that CRLNC has essentially the same computational complexity as conventional block RLNC; whereas Pace has somewhat lower computational complexity. The reduced Pace computational complexity is due to the increased number of zero-valued coding coefficients that correspond to source packets that are not considered for the coded packets early in a generation [29]. Thus, the reduced computational complexity of Pace is directly related to the lower level of protection provided to the source packets in Pace compared to conventional block RLNC and CRLNC.

We also observe from Fig. 8 that the number of operations drops with increasing code rate R. This is because an increasing code rate R, $R \to 1$, results in fewer generated coded packets, namely one coded packet for every $R/(1-R)$ source packets. On the other hand, the number of operations increases with increasing block size g, or window size w_e. The decoding complexity of block-based RLNC scales generally as $O(g^3)$ [101], [102]. With increasing g, the number of blocks (for a given number of source symbols) decreases linearly, while the computational complexity in terms of vector element operations per block increases with the third power of g [101], [102]. However, the symbol vector length also increases linearly with g. Thus, the number of symbol vector operations for a given number of source symbols increases linearly with g, as observed in Fig. 8.

For sliding window decoding, the number of insertions of received coding coefficient vectors into the receiver coding coefficient matrix $\mathbf{R}$ according to Section IV-C remains unchanged as the window size ($w_e = w_d$) grows. However, both the number of rows and the number of columns of the receiver coding coefficient matrix $\mathbf{R}$ increase linearly with the window size (cf. Section IV-F). Hence, the number of triangulation steps in the Gaussian elimination increases linearly with the window size. Similarly, the number of backward substitution steps in the Gaussian elimination, which follow sequentially after the triangulation, increases linearly with the window size. Thus, the number of symbol vector operations increases overall linearly with the window size, matching the computational complexity scaling of conventional block RLNC. (We note that the number of element operations per symbol vector operation also increases linearly with the window size.)

VI. CONCLUSION

We have proposed and evaluated a Caterpillar RLNC (CRLNC) approach with a finite encoding window and a finite decoding window. We specified the CRLNC encoding process and packet format as well as the CRLNC decoding process. CRLNC requires only minor modifications to the packet structure and encoder of conventional block based RLNC. Furthermore, CRLNC requires only low-complexity pre-processing of the received packets at the decoder; the pre-processed packets can then be fed into a standard block based RLNC decoder.

We compared CRLNC through extensive simulations with conventional block based RLNC with coded packets at the end of a block, block based RLNC with paced coded packets [29], and infinite sliding window RLNC. We found that the finite sliding window CRLNC approach combines the benefits of existing block based and infinite sliding window RLNC approaches: CRLNC achieves the low loss probabilities (i.e., high reliability) and has the low computational complexity of block based RLNC, while achieving the typically low delays of infinite sliding window RLNC. Moreover, CRLNC achieves substantially shorter delays than block based RLNC, while CRLNC does not achieve lower loss probabilities than infinite sliding window RLNC. CRLNC appears therefore well suited for streaming applications, such as VoIP and video streaming.

There are many exciting directions for future research on CRLNC. One direction is to enhance CRLNC with feedback mechanisms that could improve performance in networks that support feedback messages from the receiver to the sender [106]–[108]. Another direction is to examine the recoding of packets in network nodes, such as intermediate switches, so as to increase the resilience for subsequent network links with high packet drop probabilities [109], [110]. Packets could also be recoded in multi-hop mesh networks, where the packets of a given stream may travel over multiple paths.

ACKNOWLEDGMENT

This paper was presented in [1].

REFERENCES

[1] S. Wunderlich, F. Gabriel, S. Pandi, and F. H. P. Fitzek, "We don't need no generation—A practical approach to sliding window RLNC," in *Proc. IEEE Wireless Days*, Mar. 2017, pp. 218–223.

[2] R. Bassoli, H. Marques, J. Rodriguez, K. W. Shum, and R. Tafazolli, "Network coding theory: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1950–1978, 4th Quart., 2013.

[3] M. Z. Farooqi, S. M. Tabassum, M. H. Rehmani, and Y. Saleem, "A survey on network coding: From traditional wireless networks to emerging cognitive radio networks," *J. Netw. Comput. Appl.*, vol. 46, pp. 166–181, Nov. 2014.

[4] T. Ho *et al.*, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[5] P. Chau, T. D. Bui, Y. Lee, and J. Shin, "Efficient data uploading based on network coding in LTE-advanced heterogeneous networks," in *Proc. IEEE Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2017, pp. 252–257.

[6] H. Kang, H. Yoo, D. Kim, and Y.-S. Chung, "CANCORE: Context-aware network coded repetition for VANETs," *IEEE Access*, vol. 5, pp. 3504–3512, 2017.

[7] X. Li, Q. Chang, and Y. Xu, "Queueing characteristics of the best effort network coding strategy," *IEEE Access*, vol. 4, pp. 5990–5997, 2016.

[8] J.-S. Liu, C.-H. R. Lin, and J. Tsai, "Delay and energy tradeoff in energy harvesting multi-hop wireless networks with inter-session network coding and successive interference cancellation," *IEEE Access*, vol. 5, pp. 544–564, 2017.

[9] H. Khamfroush, D. E. Lucani, P. Pahlevani, and J. Barros, "On optimal policies for network-coded cooperation: Theory and implementation," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 2, pp. 199–212, Feb. 2015.

[10] A. Nessa, M. Kadoch, and B. Rong, "Fountain coded cooperative communications for LTE-A connected heterogeneous M2M network," *IEEE Access*, vol. 4, pp. 5280–5292, 2016.

[11] P. Pahlevani *et al.*, "Novel concepts for device-to-device communication using network coding," *IEEE Commun. Mag.*, vol. 52, no. 4, pp. 32–39, Apr. 2014.

[12] Y. Yang, W. Chen, O. Li, Q. Liu, and L. Hanzo, "Truncated-ARQ aided adaptive network coding for cooperative two-way relaying networks: Cross-layer design and analysis," *IEEE Access*, vol. 4, pp. 9361–9376, 2016.

[13] Y. Yang, W. Chen, O. Li, and L. Hanzo, "Joint rate and power adaptation for amplify-and-forward two-way relaying relying on analog network coding," *IEEE Access*, vol. 4, pp. 2465–2478, 2016.

[14] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 989–997, May 2014.

[15] N. Kumar, S. Zeadally, and J. J. P. C. Rodrigues, "QoS-aware hierarchical Web caching scheme for online video streaming applications in Internet-based vehicular ad hoc networks," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, pp. 7892–7900, Dec. 2015.

[16] M. Sipos, J. Gahm, N. Venkat, and D. Oran, "Erasure coded storage on a changing network: The untold story," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[17] L. Wang, H. Wu, and Z. Han, "Wireless distributed storage in socially enabled D2D communications," *IEEE Access*, vol. 4, pp. 1971–1984, 2016.

[18] B.-W. Chen, W. Ji, F. Jiang, and S. Rho, "QoE-enabled big video streaming for large-scale heterogeneous clients and networks in smart cities," *IEEE Access*, vol. 4, pp. 97–107, 2016.

[19] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *Proc. IEEE INFOCOM*, vol. 4. Mar. 2005, pp. 2235–2245.

[20] B. Li and D. Niu, "Random network coding in peer-to-peer networks: From theory to practice," *Proc. IEEE*, vol. 99, no. 3, pp. 513–523, Mar. 2011.

[21] M. Rekab-Eslami, M. Esmaeili, and T. A. Gulliver, "Multicast convolutional network codes via local encoding kernels," *IEEE Access*, vol. 5, pp. 6464–6470, 2017.

[22] E. Skevakis and I. Lambadaris, "Optimal control for network coding broadcast," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[23] Y. Benfattoum, S. Martin, and K. Al Agha, "QoS for real-time reliable multicasting in wireless multi-hop networks using a generation-based network coding," *Comput. Netw.*, vol. 57, no. 6, pp. 1488–1502, Apr. 2013.

[24] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. Allerton Conf. Commun. Control Comput.*, vol. 41. 2003, pp. 40–49.

[25] P. A. Chou and Y. Wu, "Network coding for the Internet and wireless networks," *IEEE Signal Process. Mag.*, vol. 24, no. 5, pp. 77–85, Sep. 2007.

[26] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and T. Larsen, "Network coding for mobile devices—Systematic binary random rateless codes," in *Proc. IEEE ICC Workshops*, Jun. 2009, pp. 1–6.

[27] D. E. Lucani, M. Médard, and M. Stojanovic, "On coding for delay—Network coding for time-division duplexing," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2330–2348, Apr. 2012.

[28] R. Prior and A. Rodrigues, "Systematic network coding for packet loss concealment in broadcast distribution," in *Proc. IEEE Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2011, pp. 245–250.

[29] S. Pandi, F. Gabriel, J. A. Cabrera, S. Wunderlich, M. Reisslein, and F. H. P. Fitzek, "PACE: Redundancy engineering in RLNC for low-latency communication," *IEEE Access*, to be published.

[30] M. Karzand, D. J. Leith, J. Cloud, and M. Médard, "Design of FEC for low delay in 5G," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 8, pp. 1783–1793, Aug. 2017.

[31] L. I. van Griethuijsen and B. A. Trimmer, "Locomotion in caterpillars," *Biol. Rev.*, vol. 89, no. 3, pp. 656–670, Aug. 2014.

[32] P. Maymounkov, N. J. A. Harvey, and D. S. Lun, "Methods for efficient network coding," in *Proc. Allerton Conf. Commun., Control, Comput.*, 2006, pp. 482–491.

[33] N. Aboutorab, P. Sadeghi, and S. Sorour, "Enabling a tradeoff between completion time and decoding delay in instantly decodable network coded systems," *IEEE Trans. Commun.*, vol. 62, no. 4, pp. 1296–1309, Apr. 2014.

[34] A. Douik, M. S. Karim, P. Sadeghi, and S. Sorour, "Delivery time reduction for order-constrained applications using binary network codes," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2016, pp. 1–6.

[35] A. Douik, S. Sorour, T. Y. Al-Naffouri, and M.-S. Alouini, "Instantly decodable network coding: From centralized to device-to-device communications," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 1201–1224, 2nd Quart., 2017.

[36] X. Li, C.-C. Wang, and X. Lin, "Optimal immediately-decodable inter-session network coding (IDNC) schemes for two unicast sessions with hard deadline constraints," in *Proc. IEEE Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2011, pp. 784–791.

[37] J. Qureshi, C. H. Foh, and J. Cai, "Online XOR packet coding: Efficient single-hop wireless multicasting with low decoding delay," *Comput. Commun.*, vol. 39, pp. 65–77, Feb. 2014.

[38] S. Sorour and S. Valaee, "Completion delay minimization for instantly decodable network codes," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1553–1567, Oct. 2015.

[39] M. Yu, N. Aboutorab, and P. Sadeghi, "From instantly decodable to random linear network coded broadcast," *IEEE Trans. Commun.*, vol. 62, no. 11, pp. 3943–3955, Nov. 2014.

[40] J. Barros, R. A. Costa, D. Munaretto, and J. Widmer, "Effective delay control in online network coding," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 208–216.

[41] R. A. Costa, D. Munaretto, J. Widmer, and J. Barros, "Informed network coding for minimum decoding delay," in *Proc. IEEE Int. Conf. Mobile Ad Hoc Sensor Syst.*, Sep./Oct. 2008, pp. 80–91.

[42] A. Fu, P. Sadeghi, and M. Médard, "Dynamic rate adaptation for improved throughput and delay in wireless network coded broadcast," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 1715–1728, Dec. 2014.

[43] P. Sadeghi, R. Shams, and D. Traskov, "An optimal adaptive network coding scheme for minimizing decoding delay in broadcast erasure channels," *EURASIP J. Wireless Commun. Netw.*, vol. 2010, Dec. 2010, Art. no. 618016.

[44] S. Sorour and S. Valaee, "An adaptive network coded retransmission scheme for single-hop wireless multicast broadcast services," *IEEE/ACM Trans. Netw.*, vol. 19, no. 3, pp. 869–878, Jun. 2011.

[45] W.-L. Yeow, A. T. Hoang, and C.-K. Tham, "Minimizing delay for multicast-streaming in wireless networks with network coding," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 190–198.

[46] D. Zeng, S. Guo, H. Jin, and V. Leung, "Segmented network coding for stream-like applications in delay tolerant networks," in *Proc. IEEE GLOBECOM*, Dec. 2011, pp. 1–5.

[47] J. Krigslund, F. Fitzek, and M. V. Pedersen, "On the combination of multi-layer source coding and network coding for wireless networks," in *Proc. IEEE Int. Workshop Comput. Aided Modeling Design Commun. Links Netw. (CAMAD)*, Sep. 2013, pp. 1–6.

[48] K. Matsuzono, H. Asaeda, and T. Turletti, "Low latency low loss streaming using in-network coding and caching," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[49] V. N. Swamy, P. Rigge, G. Ranade, A. Sahai, and B. Nikolić, "Network coding for high-reliability low-latency wireless control," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2016, pp. 1–7.

[50] K. Yu, J. Yue, Z. Lin, J. Åkerberg, and M. Björkman, "Achieving reliable and efficient transmission by using network coding solution in industrial wireless sensor networks," in *Proc. IEEE Int. Symp. Ind. Electron. (ISIE)*, Jun. 2016, pp. 1162–1167.

[51] N. dos Santos Ribeiro Júnior, R. C. Tavares, M. A. M. Vieira, L. F. M. Vieira, and O. Gnawali, "CodeDrip: Improving data dissemination for wireless sensor networks with network coding," *Ad Hoc Netw.*, vol. 54, pp. 42–52, Jan. 2017.

[52] I. Chatzigeorgiou and A. Tassi, "Decoding delay performance of random linear network coding for broadcast," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7050–7060, Aug. 2017.

[53] J. Claridge and I. Chatzigeorgiou, "Probability of partially decoding network-coded messages," *IEEE Commun. Lett.*, vol. 21, no. 9, pp. 1945–1948, Sep. 2017.

[54] G. Cocco, T. de Cola, and M. Berioli, "Performance analysis of queueing systems with systematic packet-level coding," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 4524–4529.

[55] A. Eryilmaz, A. Ozdaglar, M. Médard, and E. Ahmed, "On the delay and throughput gains of coding in unreliable networks," *IEEE Trans. Inf. Theory*, vol. 54, no. 12, pp. 5511–5524, Dec. 2008.

[56] G. Joshi, Y. Kochman, and G. Wornell. (2015). "On throughput-smoothness trade-offs in streaming communication." [Online]. Available: https://arxiv.org/abs/1511.08143

[57] M. Kwon and H. Park, "Analysis on decoding error rate of systematic network coding," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2017, pp. 258–259.

[58] X. Li, C.-C. Wang, and X. Lin, "Throughput and delay analysis on uncoded and coded wireless broadcast with hard deadline constraints," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–5.

[59] Y. Li, E. Soljanin, and P. Spasojevic, "Effects of the generation size and overlap on throughput and complexity in randomized linear network coding," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 1111–1123, Feb. 2011.

[60] M. Nistor, D. E. Lucani, T. T. V. Vinhoza, R. A. Costa, and J. Barros, "On the delay distribution of random linear network coding," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 5, pp. 1084–1093, May 2011.

[61] P. Pahlevani, S. Crisóstomo, and D. E. Lucani, "An analytical model for perpetual network codes in packet erasure channels," in *Proc. Int. Workshop Multiple Access Commun.*, vol. 10121. 2016, pp. 126–135.

[62] P. Parag and J.-F. Chamberland, "Queueing analysis of a butterfly network for comparing network coding to classical routing," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1890–1908, Apr. 2010.

[63] O. B. Rhaiem and L. Chaari, "Information transmission based on network coding over wireless networks: A survey," *Telecommun. Syst.*, vol. 65, no. 4, pp. 551–565, Aug. 2017.

[64] B. Shrader and A. Ephremides, "Queueing delay analysis for multicast with random linear coding," *IEEE Trans. Inf. Theory*, vol. 58, no. 1, pp. 421–429, Jan. 2012.

[65] A. A. Yazdi, S. Sorour, S. Valaee, and R. Y. Kim, "Optimum network coding for delay sensitive applications in WiMAX unicast," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 2576–2580.

[66] M. Yu and P. Sadeghi. (2017). "Approximating throughput and packet decoding delay in linear network coded wireless broadcast." [Online]. Available: https://arxiv.org/abs/1701.04551

[67] W. Zeng, C. T. K. Ng, and M. Médard, "Joint coding and scheduling optimization in wireless systems with varying delay sensitivities," in *Proc. IEEE SECON*, Jun. 2012, pp. 416–424.

[68] T. D. Assefa, K. Kralevska, and Y. Jiang, "Performance analysis of LTE networks with random linear network coding," in *Proc. IEEE Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May/Jun. 2016, pp. 601–606.

[69] V. Patil, S. Gupta, and C. Keshavamurthy, "An enhanced network coding based MAC optimization model for QoS oriented multicast transmission over LTE networks," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 12, pp. 843–851, Dec. 2016.

[70] A. Tassi, F. Chiti, R. Fantacci, and F. Schoen, "An energy-efficient resource allocation scheme for RLNC-based heterogeneous multicast communications," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1399–1402, Aug. 2014.

[71] S. Teerapittayanon *et al.*, "Network coding as a WiMAX link reliability mechanism: An experimental demonstration," in *International Workshop on Multiple Access Communications* (Lecture Notes in Computer Science), vol. 7642. Berlin, Germany: Springer, 2012, pp. 75–78.

[72] B. T. Swapna, A. Eryilmaz, and N. B. Shroff, "Throughput-delay analysis of random linear network coding for wireless broadcasting," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6328–6341, Oct. 2013.

[73] L. Yang, Y. E. Sagduyu, and J. H. Li, "Adaptive network coding for scheduling real-time traffic with hard deadlines," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2012, pp. 105–114.

[74] X. Zhong, Y. Qin, and L. Li, "TCPNC-DGSA: Efficient network coding scheme for TCP in multi-hop cognitive radio networks," *Wireless Pers. Commun.*, vol. 84, no. 2, pp. 1243–1263, Sep. 2015.

[75] M. Halloush and H. Radha, "Network coding with multi-generation mixing: A generalized framework for practical network coding," *IEEE Trans. Wireless Commun.*, vol. 10, no. 2, pp. 466–473, Feb. 2011.

[76] M. D. Halloush and H. Radha, "A framework for video network coding with multi-generation mixing," *J. Commun.*, vol. 7, no. 3, pp. 192–201, Mar. 2012.

[77] J. Bhatia, A. Patel, and Z. Narmawala, "Review on variants of network coding in wireless ad-hoc networks," in *Proc. IEEE Nirma Univ. Int. Conf. Eng. (NUiCONE)*, Dec. 2011, pp. 1–6.

[78] R. Liu, J. Hao, and Y. Guo, "An improved method of muti-generation mixing network coding," in *Proc. IEEE World Congr. Inf. Commun. Technol. (WICT)*, Oct./Nov. 2012, pp. 1086–1091.

[79] A. Heidarzadeh and A. H. Banihashemi, "Overlapped chunked network coding," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Jan. 2010, pp. 1–5.

[80] A. Heidarzadeh and A. H. Banihashemi, "Analysis of overlapped chunked codes with small chunks over line networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul./Aug. 2011, pp. 801–805.

[81] J. Cloud and M. Médard, "Network coding over SATCOM: Lessons learned," in *Proc. Int. Conf. Wireless Satellite Syst.*, 2015, pp. 272–285.

[82] J. Cloud and M. Médard, "Multi-path low delay network codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.

[83] A. Garcia-Saavedra, M. Karzand, and D. J. Leith, "Low delay random linear coding and scheduling over multiple interfaces," *IEEE Trans. Mobile Comput.*, to be published.

[84] M. Karzand and D. J. Leith, "Low delay random linear coding over a stream," in *Proc. 52nd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2014, pp. 521–528.

[85] J. Cloud, D. Leith, and M. Médard, "A coded generalization of selective repeat ARQ," in *Proc. IEEE INFOCOM*, Apr./May 2015, pp. 2155–2163.

[86] E. Drinea, L. Keller, and C. Fragouli, "Real-time delay with network coding and feedback," *Phys. Commun.*, vol. 6, pp. 100–113, Mar. 2013.

[87] Y. Lin, B. Liang, and B. Li, "SlideOR: Online opportunistic network coding in wireless mesh networks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–5.

[88] J. K. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," *Proc. IEEE*, vol. 99, no. 3, pp. 490–512, Mar. 2011.

[89] P.-U. Tournoux, A. Bouabdallah, J. Lacan, and E. Lochin, "On-the-fly coding for real-time applications," in *Proc. ACM Int. Conf. Multimedia*, 2009, pp. 889–892.

[90] T. T. Thai, E. Lochin, and J. Lacan, "Online multipath convolutional coding for real-time transmission," in *Proc. IEEE Int. Packet Video Workshop (PV)*, May 2012, pp. 41–46.

[91] J. Detchart, E. Lochin, J. Lacan, and V. Roca, "Tetrys, an on-the-fly network coding protocol," HAL Inria, Tech. Rep. hal-01089745v3, 2015.

[92] V. Roca, B. Teibi, C. Burdinat, T. Tran-Thai, and C. Thienot, "Block or convolutional AL-FEC codes? A performance comparison for robust low-latency communications," HAL Inria, Tech. Rep. hal-01395937v2, 2017.

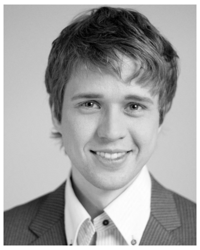
[93] D. Gligoroski, K. Kralevska, and H. Øverby, "Minimal header overhead for random linear network coding," in *Proc. IEEE Int. Conf. Commun. Workshop (ICCW)*, Jun. 2015, pp. 680–685.

[94] N. Thomos and P. Frossard, "Toward one symbol network coding vectors," *IEEE Commun. Lett.*, vol. 16, no. 11, pp. 1860–1863, Nov. 2012.

[95] J. B. Fraleigh and R. A. Beauregard, *Linear Algebra*, 3rd ed. London, U.K.: Pearson, 2013.

[96] M. V. Pedersen, J. Heide, and F. H. P. Fitzek, "Kodo: An open and research oriented network coding library," in *NETWORKING 2011 Workshops* (Lecture Notes in Computer Science), vol. 6827. Berlin, Germany: Springer, 2011, pp. 145–152.

[97] D. Chkliaev, J. Hooman, and E. de Vink, "Verification and improvement of the sliding window protocol," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2003, pp. 113–127.
[98] L. Lockefeer, D. M. Williams, and W. Fokkink, "Formal specification and verification of TCP extended with the window scale option," *Sci. Comput. Program.*, vol. 118, pp. 3–23, Mar. 2016.
[99] A. U. Shankar, "Verified data transfer protocols with variable flow control," *ACM Trans. Comput. Syst.*, vol. 7, no. 3, pp. 281–316, 1989.
[100] N. V. Stenning, "A data transfer protocol," *Comput. Netw.*, vol. 1, no. 2, pp. 99–110, Sep. 1976.
[101] H. Shin and J.-S. Park, "Optimizing random network coding for multimedia content distribution over smartphones," *Multimedia Tools Appl.*, vol. 76, no. 19, pp. 19379–19395, Oct. 2017.
[102] S. Wunderlich, J. A. Cabrera, F. H. P. Fitzek, and M. Reisslein, "Network coding in heterogeneous multicore IoT nodes with DAG scheduling of parallel matrix block operations," *IEEE Internet Things J.*, vol. 4, no. 4, pp. 917–933, Aug. 2017.
[103] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell Syst. Tech. J.*, vol. 39, no. 5, pp. 1253–1265, Sep. 1960.
[104] E. O. Elliott, "Estimates of error rates for codes on burst-noise channels," *Bell Syst. Tech. J.*, vol. 42, no. 5, pp. 1977–1997, Sep. 1963.
[105] P. Garrido, D. E. Lucani, and R. Agüero, "Markov chain model for the decoding probability of sparse network coding," *IEEE Trans. Commun.*, vol. 65, no. 4, pp. 1675–1685, Apr. 2017.
[106] M. Esmaeilzadeh, N. Aboutorab, and P. Sadeghi, "Joint optimization of throughput and packet drop rate for delay sensitive applications in TDD satellite network coded systems," *IEEE Trans. Commun.*, vol. 62, no. 2, pp. 676–690, Feb. 2014.
[107] A. Moreira, L. Almeida, and D. E. Lucani, "Merging network coding with feedback management in multicast streaming," *ACM SIGBED Rev.*, vol. 12, no. 3, pp. 49–52, Jun. 2015.
[108] C. W. Sung, K. W. Shum, L. Huang, and H. Y. Kwan, "Linear network coding for erasure broadcast channel with feedback: Complexity and algorithms," *IEEE Trans. Inf. Theory*, vol. 62, no. 5, pp. 2493–2503, May 2016.
[109] A. J. Aljohani, S. X. Ng, and L. Hanzo, "Distributed source coding and its applications in relaying-based transmission," *IEEE Access*, vol. 4, pp. 1940–1970, 2016.
[110] G. Giacaglia, X. Shi, M. Kim, D. E. Lucani, and M. Médard, "Systematic network coding with the aid of a full-duplex relay," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 3312–3317.

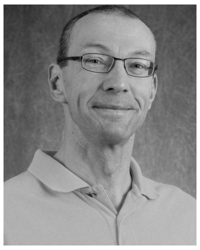


SIMON WUNDERLICH received the Dipl.-Inf. degree in computer science from Chemnitz Technical University, Germany, in 2009. He is currently pursuing the Ph.D. degree in electrical engineering with the Technical University of Dresden, Germany. He has co-authored the Wi-Fi mesh software B.A.T.M.A.N. advanced.

FRANK GABRIEL received the Dipl.-Inf. degree in computer science from the Technical University Chemnitz, Germany, in 2011. He is currently a Ph.D. Researcher with the Deutsche Telekom Chair of Communication Networks, TU Dresden.

SREEKRISHNA PANDI was born in Chennai, India. He received the B.E. degree in electronics and instrumentation engineering from Anna University in 2013 and the master's degree in nanoelectronic systems from the Technical University of Dresden (TU Dresden) in 2015. He has been a Ph.D. Researcher with the Deutsche Telekom Chair of Communication Networks, TU Dresden, since 2015.

FRANK H. P. FITZEK received the Diploma (Dipl.Ing.) degree in electrical engineering from RWTH Aachen University, Germany, in 1997, and the Ph.D. (Dr.-Ing.) degree in electrical engineering from the Technical University of Berlin, Germany, in 2002. He became an Adjunct Professor with the University of Ferrara, Italy, in 2002. In 2003, he joined Aalborg University as an Associate Professor and later became a Professor. He co-founded several start-up companies starting with acticom GmbH, Berlin, in 1999. He is currently a Professor and the Head of the Deutsche Telekom Chair of Communication Networks, Technical University of Dresden, Germany, and also coordinating the 5G Laboratory, Germany. His current research interests are in the areas of wireless and mobile 5G communication networks, mobile phone programming, network coding, cross layer and energy efficient protocol design, and cooperative networking. He was selected to receive the NOKIA Champion Award several times in a row from 2007 to 2011. In 2008, he received the Nokia Achievement Award for his work on cooperative networks. In 2011, he received the SAPERE AUDE research grant from the Danish Government. In 2012, he received the Vodafone Innovation prize. In 2015, he received the honorary degree Doctor Honoris Causa from the Budapest University of Technology and Economy.



MARTIN REISSLEIN (S'96–A'97–M'98–SM'03–F'14) received the Ph.D. degree in systems engineering from the University of Pennsylvania in 1998. He is currently a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe. He chairs the steering committee of the IEEE Transactions on Multimedia. He currently serves as an Associate Editor of the IEEE Transactions on Mobile Computing, the IEEE Transactions on Education, the IEEE Access, and *Computer Networks* and *Optical Switching and Networking*. He is an Associate Editor-in-Chief of the IEEE Communications Surveys and Tutorials.