On the Impact of Coding Depth in Sliding Window Random Linear Network Coding Schemes

Foteini Karetsi*, Christos Liaskos*†, Sotiris Ioannidis†‡ and Evangelos Papapetrou*
*University of Ioannina, Greece, †Foundation for Research and Technology - Hellas (FORTH),
‡Technical University of Crete, Greece
Email: [fkaretsi, cliaskos, epap]@cs.uoi.gr, sotiris@ece.tuc.gr

***Abstract*—Sliding Window Random Linear Network Coding (RLNC) offers a clear path towards achieving ultra-high reliability and low latency at the same time. Such requirements are pivotal for a wide range of applications in the future Internet as well as in 5G and beyond networks. While traditional RLNC has been extensively used for some years now, its Sliding Window flavor is rather recent and extremely promising because of its implementation advantages and the high degree of customization. Probably the most essential parameter of Sliding Window RLNC is the coding depth, i.e., the extent of non-coded packets protected by a coded one. In this work, for the first time, we elaborate on properly choosing the coding depth and shed light on the related trade-offs. We, first, show, experimentally, that significant performance gains can be obtained by fine-tuning the coding depth. Then, we propose and validate an analytical framework that allows us to decide the coding depth based on a channel's reliability profile. Finally, we introduce a dynamic algorithm that, based on our analytical findings, can improve the performance of sliding window RLNC in the presence of bursts of errors.**

***Index Terms*—sliding window RLNC, Ultra Reliable Low-Latency Communication (URLLC), Random Linear Network Coding (RLNC)**

I. Introduction

Ultra-reliable Low Latency Communication (URLLC) [1] is envisioned to support a plethora of mission-critical applications in 5G and beyond networks, which pose stringent requirements in terms of reliability and latency. In order to perform efficiently, various domains of applications, such as immersive entertainment platforms, e.g., Virtual Reality (VR) and Augmented Reality (AR) systems, autonomous vehicles, smart industries and healthcare, mandate block error rates that typically range between 10^{-9} and 10^{-5} [2] while targeting for end-to-end latency up to $1ms$, similar to the URLLC specification. Towards achieving these goals, various sophisticated techniques are deployed to utilize more available spectrum and deliver information with high-data rates, e.g. techniques related to millimeter-wave (mmWave) communication. Unfortunately, those techniques focus on bandwidth and do not deal with the severe losses observed in unreliable media. Hence, mechanisms for tackling packet losses are a necessity either in the data link or transport layer.

Although several legacy techniques, e.g., Automatic Repeat ReQuest (ARQ) and Forward Error Correction (FEC) schemes, have been deployed to mitigate packet losses in unreliable channels, they struggle to meet the stringent requirements specified by URLLC. Lately, RLNC schemes have been successfully examined as an efficient reliability mechanism, yet, there is still room for improvement. Among other variants, sliding window RLNC schemes demonstrate promising performance towards achieving the URLLC reliability-delay trade-off. Such methods exploit a dynamically changing group of non-coded packets, i.e., the *sliding window*, during the coding process, which provides flexibility and improved performance. To further enhance reliability, some sliding window RLNC schemes may employ re-transmissions at the expense of increased delay. In fact, sliding window RLNC is quite attractive to various fields, such as multimedia [3]–[5] or IoT scenarios [6] while they have also been examined in the context of transport-layer protocols, initially regarding the TCP protocol [7] and, more recently, QUIC [8], as reliability mechanisms.

While sliding window RLNC has proved to be an enabler of URLLC, little attention has been paid to the optimization of the coding efficiency of such schemes. In particular, a critical parameter for the performance of sliding window RLNC schemes is the number of non-coded packets involved in the encoding process, also known as the *coding window size*. In our previous work [9], [10], we facilitated the discussion about the coding window size by introducing the concept of *coding depth*, which refers to the range of non-coded packets protected by a redundant coded one. A similar concept is also proposed in [6]. However, to the best of our knowledge, there is no previous work highlighting the impact of coding depth on the throughput-delay vs complexity trade-off. Additionally, the optimal determination of coding depth is still an open question. *We make the observation that an appropriate selection of the coding depth is of high importance for the related trade-offs and can produce significant performance improvements.* This is extremely important for operating in channels with diverse error profiles and in channels with variable error rates, where the coding scheme should be able to detect the level of packet errors and appropriately adjust its coding parameters in order to achieve an optimal performance-complexity trade-off.

In this work, our primary goal is to investigate the impact of coding depth on the performance of sliding window RLNC protocols. Our main contributions are:

- We experimentally show that the appropriate selection of coding depth can provide significant performance improvements without the need to increase the coding redundancy, which comes at the high cost of bandwidth

consumption.

- We propose an analytical method to approximate a coding depth value that produces near-optimal performance. More specifically, the analysis takes into account the channel's reliability properties to provide a coding depth value that improves the throughput-delay vs complexity trade-off of the coding scheme.
- We devise an efficient algorithm for the dynamic adaptation of coding depth in the presence of bursts of errors in the channel. The algorithm is tailored for protocols where the coding and sliding windows do not coincide. Through this algorithm, we manage to further reduce the overall protocol complexity during idle periods, i.e., when few or no errors are detected in the channel.
- We experimentally confirm, through extensive simulations, the effectiveness of the proposed solutions. More important, we validate our analytical model and confirm that the appropriate selection of coding depth can produce a noticeable reduction in complexity, especially in the case of bursts.

The rest of this paper is organized as follows. Section II provides the basic concepts of Sliding Window RLNC and a review of the protocols in this class. In Section III, we argue that the appropriate selection of coding depth size may significantly enhance the coding operation without increasing coding redundancy. We back up our arguments through an extensive experimental analysis. In section IV, we detail the analytical framework to estimate the optimal coding depth value for specific channel conditions followed by the experimental evaluation of this approach in section V. We present an adaptive algorithm for coping with bursts of errors and assess its performance in section VI. Section VII summarizes our findings and provides a list of future extensions of this work.

II. Background and Related Work

A. A Sliding Window RLNC Primer

RLNC relies on the generation of random coefficients in a given finite field (F_{2^s}) in order to obtain random linear combinations of the uncoded packets, called *native or source packets* [11]. The generated *coded* packets will be utilized by the receiver to recover any lost source packet. It is proved that the probability of generating linearly dependent packets depends on the field size and becomes negligible for large field sizes [12]. Hence, we assume a sufficiently large finite field, such as F_{2^8}, to avoid linear dependencies of the encoded packets. As for the decoding process, the receiver needs to solve a linear system by performing Gauss-Jordan elimination or other decoding optimizations proposed in literature, e.g. [13]–[16], in order to retrieve the original packets. Decoding entails the use of a *decoding matrix* D, which is populated by *innovative* packets, i.e., the ones that increase the rank of the matrix.

Unlike traditional RLNC schemes, i.e., block-based ones [13], [17]–[25], where encoding uses a fixed-size group of source packets, sliding window approaches are quite more flexible and efficient [3]–[10], [16], [26]–[39]. In such methods, coded packets are created as random linear combinations

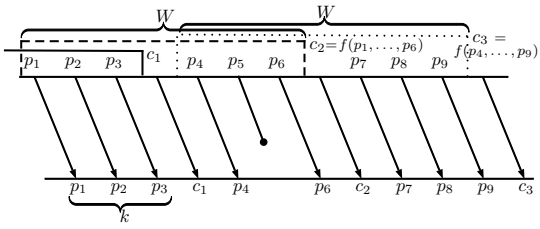


Fig. 1. Systematic sliding window RLNC example ($W = 6$, $d = 2$ and $R{=}3/4$, $k{=}3$, $r{=}1$). Dashed and dotted lines indicate the coding window contents at the time of c_2's and c_3's generation, respectively.

of a dynamic set of packets, known as the sliding (or coding) window, where source packets may be added or removed at any time. In systematic Sliding Window RLNC, which is the most prominent example of this class, the source sends out k native packets before transmitting r coded (usually $r = 1$) that provide protection to the k native. Two major parameters affect the coding performance of sliding window RLNC, that is the level of redundancy introduced into the original data stream, determined by the *code rate* R, and the *coding window size* (W). The former is expressed as $R = \frac{k}{k+r}$. On the contrary, the coding window size delineates the number of source packets involved in generating a coded one and, therefore, are protected through coding. Fig. 1 illustrates the operation of systematic Sliding Window RLNC. To facilitate the discussion regarding the impact of the coding window size (W), we introduced the concept of *coding depth* (d) in [9], [10]. Assuming a specific code rate R, coding depth describes the number of k-packet groups used in constructing a coded packet. In other words, the coding depth quantifies the number of source packets "covered" by a coded one, i.e., it captures the level of protection the coding scheme offers to the source packets. Note that, the coding window size W can be expressed as the product of coding depth and k, i.e., $W = d \times k$.

B. Review of Sliding Window RLNC schemes

Existing sliding window RLNC schemes [3]–[10], [16], [26]–[38] could be categorized according to various classification rules, as illustrated in Fig. 2. In particular, sliding window RLNC protocols can be classified, based on the "per-packet" coding approach, into *full-vector* [28], [29] schemes, where each transmitted packet is a coded one, and *systematic* ones [40], where coding is applied only on the redundant packets. The latter allows the transmission of a mixture of source and coded packets, thereby achieving lower mean packet delivery delay compared to full-vector codes [10]. Hence, the majority of the existing sliding window RLNC schemes adopt systematic codes. Employing feedback to trigger retransmissions in a similar fashion as in ARQ protocols [5], [7], [9], [10], [15], [16], [28]–[33] is commonly incorporated in sliding window RLNC. In such schemes, unless redundant coded packets suffice for loss recovery, feedback activates the transmission of additional packets (coded or not) to facilitate the decoding process and increase the probability of packet recovery. On the contrary, schemes that rely exclusively on coded redundancy to recover lost packets [34] can be also

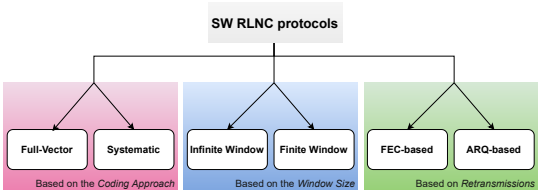


Fig. 2. A classification of existing sliding window RLNC protocols.

considered to fall into the category of *FEC-based* schemes. The latter leverage feedback, if available, to solely update the coding window's contents [3], [4].

As mentioned earlier, probably the most critical parameter for the performance of sliding window RLNC is the *coding window size*, i.e., the number of source packets included in the coding process. An elementary, yet computationally prohibitive, approach entails the use of an *infinite window* [26], where all previously transmitted source packets are used in the creation of a new coded one, that is they are never removed from the coding window. Most algorithms, though, follow a *finite* window approach [3], [5], [9], [10], [16], [34] and set an upper bound in the number of source packets existing in the sliding window. Nevertheless, they commonly neglect to elaborate further on the reasoning behind choosing a specific window size. Typically, in FEC-based schemes, the coding window size is somehow arbitrarily defined or the protocols are tested for different values. On the other hand, in ARQ-based sliding window RLNC [5], [7], [15], [16], [28]–[32], the determination of the coding window size is associated with the data flow process, i.e., based on the bandwidth-delay product. This practice, though, poses limitations in the performance of sliding window RLNC protocols, as we identified in [9], [10], where we argue that coding parameters should be chosen based on the link loss profile and not the bandwidth-delay product.

While we empirically examined the problem of correctly choosing the coding window size (or equivalently, as we explained previously, the coding depth) and its impact on sliding window RLNC protocols' performance in [9], [10], the actual method for calculating a value is still an open question. Additionally, a static configuration of RLNC coding parameters is far from optimal when bursts are present in the channel and increased protection is required to tackle packet losses. A common approach proposed in the literature for coping with loss variations and increase robustness focuses on dynamically adapting the code rate R, i.e., increase the level of redundancy [19]–[25], [27]–[29]. While rate adaptation seems to be an appropriate solution, it comes at the high cost of bandwidth consumption required for transmitting more redundant information. On the contrary, we make the observation that dynamically choosing the coding depth outweighs rate adaptation in that it does not require additional bandwidth. One can see our approach as an analogy to the existing concept of adjusting the block size in traditional block-based RLNC schemes [18]–[20]. However, implementing the strategy in the context of a sliding window is an equivalently challenging problem that is yet to be investigated.

Concluding, in this work, we thoroughly investigate the impact of an appropriate coding depth on the performance of sliding window RLNC. In the following, we first focus on the comparison between rate adaptation and coding depth adjustment and back up our arguments with experimental data. Then, we focus on systematically choosing a suitable coding depth value and investigate how to cope with bursts of errors.

TABLE I
EXPERIMENTAL SETUP: MAIN PARAMETERS

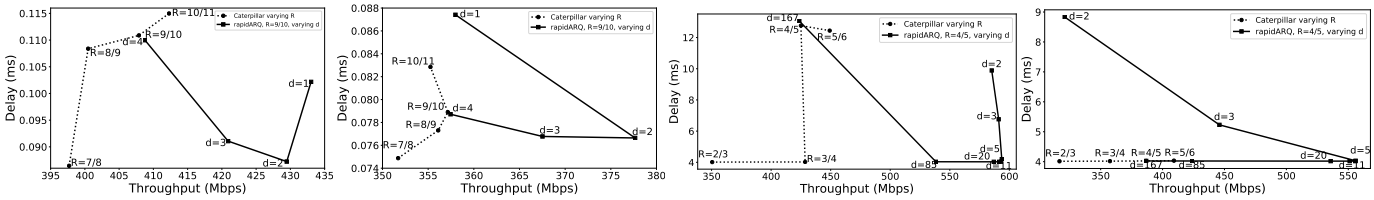
Parameter	*5G link*	*Satellite link*
Propagation delay	66.7μs	4ms
Bandwidth	500Mbps	1Gbps
Packet error rate (p_l) (one-way)	5%	10%
Sliding Window Size (W)	43	668
Packet Size	200 bytes	1500 bytes
Traffic Type	CBR	CBR

III. RATE ADAPTATION VS CODING DEPTH

As we previously discussed, the common approach in the related literature is to adapt the code rate in order to cope with increased packet losses. Nonetheless, this comes at the cost of bandwidth consumption because more redundant packets need to be transmitted. *On the contrary, we make the observation that adjusting the coding depth d can improve the resilience to packet errors without posing additional bandwidth overhead.* This is because we continue sending out the same number of redundant packets. Should an adequate coding depth be in use, we argue that conventional sliding window RLNC schemes can improve their efficiency. We expect that the importance of the optimal adaptation of d is even greater under dynamic channel conditions, particularly when source packets require increased protection. The trade-off is a slight overhead in the coding complexity that can be negligible if appropriate coding depth management is used.

To validate our observation, we evaluate the throughput-delay performance for different values of d with respect to the case that the we use different code rates (R). To this end, we use Caterpillar RLNC-FB [16] as one of the most representative ARQ-based sliding window RLNC protocol and rapidARQ [9] as the first protocol of this class that allows setting the coding depth. We simulate both protocols in ns2 [41]. The comparison results can easily be reproducible in ns3 simulator [42] after transferring the protocols' implementation, since the underlying link and error models are directly ported from ns2 to ns3. For the comparison, we adopt the two different scenarios proposed in [10]. The first refers to a typical 5G [43] terrestrial wireless link whereas the second one corresponds to a high-speed link to a low-earth orbiting satellite [44]. Table I summarizes the parameters used in our evaluation for each test scenario. Without loss of generality, we assume that channel's losses are uniformly distributed in time, however loss variations do exist. We examine bursty channels in the following.

We organize the experiment as follows. For rapidARQ we set R according to the link's loss rate (p_l), so that, on average, the coding redundancy is sufficient for coping with the average number of expected packet losses. We herein exploit the typical scenario of transmitting $r = 1$ redundant coded packet,



(a) (b) (c) (d)

Fig. 3. Throughput-delay performance of Caterpillar RLNC-FB [16] for different R values and rapidARQ [9] for different d values: a),c) 5G and satellite scenarios without re-transmissions, and b),d) 5G and satellite scenarios with up to $RTX = 3$ re-transmissions.

although the implementation can be generalized to larger r values. More specifically, we choose $R = 9/10$ in the 5G scenario and $R = 4/5$ in the satellite case. Then, we vary the coding depth between $[1, \lfloor \frac{W}{k} \rfloor]$, where W is the maximum size of the coding window allowed by the channel. For the 5G scenario, we illustrate all possible values, namely $d \in [1, 4]$, whereas in the satellite scenario, we choose to illustrate indicative values, mainly those yielding the best throughput-delay performance as well as the ones corresponding to the upper and lower bounds of d (i.e., $2, 3, 5, 7, 20, 85, 167$). For Caterpillar RLNC-FB, again, we use the optimal R configuration (based on p_l, similar to rapidARQ). However, since Caterpillar RLNC-FB does not allow customizing the coding depth (or equivalently the coding window size), we examine the protocol's performance for other values of R (both smaller and greater). In other words, we study the cases with more or less redundancy.

Figures 3(a) and 3(b) illustrate the throughput-delay performance of Caterpillar RLNC-FB and rapidARQ in the 5G scenario. At first sight, rapidARQ outweighs Caterpillar for every value tested. Regarding Caterpillar, we can observe that larger redundancy than the basic value of $R = 9/10$ (i.e., $R = 7/8$ and $R = 8/9$) results in poorer throughput performance. This is expected since more redundancy consumes more bandwidth. However, the larger the redundancy, the smaller the average delay is, since redundant coded packets are transmitted more often and this allows the receiver to quickly recover any missing source packets. The advantage in average delay dissipates, though, when re-transmissions are included (Fig. 3(b)), since they mostly account for packet recovery at the receiver side. By accepting a slight overhead in average delay, we can apply a scarcer code rate, e.g., $R = 9/10$, which results in improved throughput performance. On the contrary, rapidARQ proves that it performs optimally simply by adjusting d and without modifying R. While in Caterpillar increased redundancy aims to enhance protection of source packets, increased protection and superior overall performance can be jointly achieved by only adapting d. Not only does rapidARQ accomplish comparable average delay with $R{=}7/8$ with and without re-transmissions, but also it outperforms Caterpillar RLNC-FB massively regarding throughput.

Similar results are obtained in the satellite scenario, too. Figures 3(c) and 3(d) highlight the superior throughput-delay performance when adjusting the coding depth (rapidARQ) instead of the code rate (Caterpillar RLNC-FB). Here, the throughput benefits are quite more evident, depending on the applied value of R in Caterpillar RLNC-FB. In case of high redundancy ($R = 2/3$), the throughput benefits can reach up to 69% and 75% for $RTX = 0$ and $RTX = 3$, respectively. For the default code rate ($R = 4/5$), the net increase in throughput runs to $\sim 169Mpbs$ with and without re-transmissions, which corresponds to around 40% increase with the same introduced redundancy. As for the average delay, rapidARQ achieves a minimum delay equivalent to the one noticed when $R = 2/3$ is used in Caterpillar RLNC-FB. RapidARQ's performance can be considered poor only in very small values of d, such as $d = 2$, in the case that re-transmissions are used. This is reasonable because such small values of d correspond to minimal coding protection. Thus, most packet losses are handled by re-transmissions which increases delay. Without re-transmissions, though, Caterpillar RLNC-FB substantially falls short in delay, since it introduces more overhead, through the increased code rate R, to cope with packet losses.

The previous comparison indicates that the advantages of rate adaptation come at a high throughput cost. On the contrary, by altering only the coding depth d, rapidARQ demonstrates an improved performance. Equally noteworthy is the fact that a wide range of d values can perform efficiently. Actually, rapidARQ's performance degrades only for very small or large values of d. Note that, in fig. 3(c) and 3(d), when d ranges in $[5, 20]$, rapidARQ achieves almost optimal performance. The improvement is notable against Caterpillar RLNC-FB even when $d = 85$. However, the optimal range of values for d depends on the test scenario and has yet to be determined. In the following section we tackle this problem.

IV. Choosing the Appropriate Coding Depth

In the previous section, we experimentally showed that setting an appropriate coding depth does result in superior performance. However, an efficient method for estimating a suitable d value based on the channel's characteristics has yet to be examined. At this point we should stress that the impact of d on a protocol's performance heavily depends on the decoding policies implemented at the receiver to keep the decoding matrix size at a reasonable size. In the following analysis we assume that the receiver does not wait for a possible decoding after receiving d coded packets for a lost non-coded packet. This assumption is based on the reasonable strategy not to wait for a possible decoding beyond that point because the probability of decoding is small and the trade-off in terms of delay and decoding matrix size is not negligible. However, we believe that the following analysis can be used

as a starting point for delineating the performance of Sliding Window RLNC protocols with different decoding policies.

As we observed in section III, by managing the coding depth we can achieve increased robustness in conjunction with enhanced throughput-delay performance under variable packet loss rates. This is because fine-tuning the coding depth makes possible the maximization of coding benefits. Interestingly enough, we can achieve near-optimal performance not for a single d value but for a wide range of values (see, for example, fig. 3(c) and 3(d)). This is a reasonable result since increasing d also increases the offered protection. When d goes beyond a certain value no significant improvement is witnessed. We call this value the threshold value of d (d_{th}). However, depending on the channel properties and loss profile, it is not always the case that a wide range of d values can produce a near optimal performance. For example, this is the case of the 5G scenario discussed in the previous section. As can be seen in fig. 3(c) and 3(d), the range of d values that optimize performance is rather narrow (i.e., $d = 2, 3$). The reason is that, while d should be greater than d_{th}, it also should not exceed an upper value d_{up}. In other words,

$$d_{th} \leq d \leq d_{up} \tag{1}$$

The existence of d_{up} was discussed in our previous work [10]. This upper limit is imposed by the limited sliding window size at the receiver side. To elaborate on this, let us examine Fig. 4 where a series of coding cycles (i.e., a set of non-coded packet followed by a coded one) at the sender is presented. As can be seen, when a coded packet arrives at the destination, the decoding process involves all the native packets within the coding window, i.e., the $d \cdot k$ last native packets, and the corresponding coded ones in this range. However, note that the oldest coded packet in this group of packets also "contains" the previous $d \cdot k$ native packets. As a result, in the worst case, in any given moment, the last $(2d-1) \cdot k$ packets are involved in the decoding process at the receiver. However, the receiver has a limited sliding window size W. Therefore, $(2d-1) \cdot k \leq W$, otherwise the receiver will start dropping packets that are outside the sliding window although those packets may be necessary for the decoding process to complete. The previous limitation results in

$$d \leq \frac{1}{2}(\lfloor \frac{W}{k} \rfloor + 1) = d_{up} \tag{2}$$

Clearly, d_{up} depends on the channel characteristics since W is set based on the channel's bandwidth-delay product and k is chosen based on the packet error rate.

It is straightforward from the previous discussion that, depending on the channel characteristics, it may either be that $d_{up} \leq d_{th}$ or $d_{th} \leq d_{up}$. It is reasonable that the former inequality holds in channels with small bandwidth-delay products and/or relatively large loss rates while the latter is valid in channels with large bandwidth-delay products and rather small loss rates. By combining the former inequalities with (1), the optimal value for d can be determined as

$$d_{opt} = min\{d_{th}, d_{up}\} \tag{3}$$

This equation can explain the behavior of rapidARQ witnessed in the experiments of Section III. In the case of the 5G scenario, the bandwidth delay product is rather small therefore the best value for d is dominated by d_{up}. The opposite is true for the satellite scenario where the large bandwidth-delay product indicates that $d_{opt} = d_{th}$. Keep in mind that, in this case, rapidARQ's throughput-delay performance is near optimal for all values in the range $[d_{th}, d_{up}]$ since all those values provide the required protection.

While calculating d_{up} is straightforward, determining d_{th} is a daunting challenge because it depends on a variety of coding/decoding design choices. Not surprisingly, to the best of our knowledge, such an analytical calculation does not exist in literature. Our approach is to derive an analytical framework (see Section IV-A) that will allow us to obtain a rough estimation ($\tilde{d}_{th}$) of d_{th}. In the case that $\tilde{d}_{th} > d_{up}$ clearly the best choice for d is d_{up}. However, when $\tilde{d}_{th} < d_{up}$, our estimation will produce a near optimal performance if $\tilde{d}_{th} \in [d_{th}, d_{up}]$.

A. Analysis of the Impact of Coding Depth

In this section we provide the theoretical analysis for computing a rough estimation of d_{th}. As previously explained, when utilizing the idea of coding depth, the receiver side can perceive the packet reception/decoding process as consisting of a repetition of *coding cycles*. Each coding cycle consists of k native (or uncoded) packets and a coded one. The coded packet in a specific coding cycle is built based on the native packets comprising the last d coding cycles, including the current one. If the receiver misses native packets (because they are lost) then, upon the reception of a coded packet, checks whether decoding the lost packets is possible. Let us assume that at time $t = 0$ the decoder is not missing any packet and a new coding cycle starts where at least one native packet is missing. The decoder will try to recover the lost packet(s) at the end of the decoding cycle, i.e., after receiving redundant information in the form of a coded packet. If this is not possible, the process will continue with another chance to recover the packet(s) in the second coding cycle. Note that, in the meanwhile, the number of missing packets may increase if native packets belonging to the second coding cycle are missing. The process will continue in the following coding cycles until decoding (the entirety of the missing packets) is possible or the packets are confidently deemed unrecoverable. For the latter, as we previously stated, we assume that this happens after d cycles because, beyond that point, no more coded packets containing the lost packets in round 1 will be received, thus the probability of decoding beyond that point is

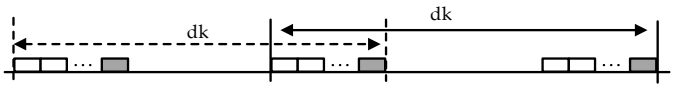


Fig. 4. Successive coding cycles observed at the sender side in a SW RLNC protocol. The product $d \cdot k$ describes the coding window size.

small and the trade-off in terms of delay and decoding matrix size is not negligible.

We wish to calculate the probability that the receiver will not to be able to recover (through decoding) the missing packets after i coding cycles. Let $\widetilde{P}_i$ denote this probability. We assume that X_i denotes the random variable corresponding to the number of lost packets (either native or coded) during coding cycle i. Note that X_i is binomially distributed, i.e., $X_i \sim B(k+1, p)$ where $k+1$ is the number of packets (native and coded ones) in the coding cycle and p is the channel's loss rate. Let also W_i denote the opposite of the decoding matrix's rank deficiency, i.e., the difference between the decoding matrix's rank and its number of rows (number of linearly independent packets), at the end of cycle i. In other words, W_i represents the number of linearly independent packets needed by the receiver to perform a decoding. Note that:

$$W_i = max(0, W_{i-1} + X_i - 1), \ \ \forall\, i > 0, \ W_0 = 0 \qquad (4)$$

The previous formula is easy to understand if we keep in mind that there are two possibilities for the X_i packets missing during cycle i. The first is that all X_i packets are native and the one coded packet offers a linearly independent combination containing those packets therefore it can increase the rank of the decoding matrix. The second possibility is that only $X_i - 1$ packets are native and the one coded packet is lost. In this case, again, W increases by $X_i - 1$. It is clear that $W_i = 0$ implies that decoding is possible at round i. Similarly, no decoding is possible after i coding cycles if and only if $W_i > 0$ and $W_j > 0 \,\forall\, j < i$. Therefore, we can write $\widetilde{P}_i$ as:

$$\widetilde{P}_i = \sum_{x=1}^{ik} P\{W_i = x \mid W_j > 0, \forall\, j < i\} \qquad (5)$$

Note that the maximum value of W_i is $i \cdot k$. This can be derived by (4) if $X_j = k+1, \ \forall\, j$, which corresponds to the worst case where all native and all coded packets are lost in every coding cycle. Given $\tilde{P}_i$, we can write the probability of decoding in any of the cycles up to i as:

$$\hat{P}_i = 1 - \widetilde{P}_i \qquad (6)$$

Moreover, we can prove that:

Lemma 1. *The probability of decoding at cycle i, i.e., without any decoding at a previous round, is:*

$$P_i = 1 - \widetilde{P}_i - \hat{P}_{i-1} \qquad (7)$$

Proof. By its definition, P_i can be written as $P_i = P\{W_i = 0 \mid W_j > 0, \forall\, j < i\}$. Thus, we can derive P_i by examining all sequences of $W_1, W_2, \ldots W_{i-1}, W_i (= 0)$ and eliminating those containing at least one 0 in any place $j \leq i - 1$ because those latter sequences correspond to chains of events where a decoding occurs before cycle i. Equivalently, we first examine all possible sequences of $W_1, W_2, \ldots W_{i-1}, W_i$. The probability of any such sequence occurrence is clearly 1. Then we exclude those sequences that contain at least 0 in any place $j \leq i - 1$. Note that, by definition, the probability that any such sequence takes place is $\hat{P}_i$. Finally, from the remaining set of sequences, each of which does not contain any 0 in any position $j \leq i-1$, we exclude the ones that end with $W_i > 0$. In other words, we exclude that sequences that end in 0 but do not not contain any other 0s. The probability of any such sequence taking place is, by definition, $\widetilde{P}_i$. □

Going back to (5), clearly, it is easy to derive $\widetilde{P}_i$ if we can analytically compute $P_x^{(i)} = P\{W_i = x \mid W_j > 0, \forall\, j < i\}$, i.e., the probability that the receiver's decoding matrix requires x linearly independent packets at cycle i while no decoding has occurred in the previous rounds. To this end, as a first step, we use (4) recursively to re-write $P_x^{(i)}$. More specifically, when $W_j = 0 \,\forall\, j < i$, we can conclude from (4) that $W_i = X_1 + \cdots + X_i - i$. As a result, $P_x^{(i)}$ can be re-written as:

$$P_x^{(i)} = P\{X_1 + \cdots + X_i = x + i \mid W_j > 0, \forall\, j < i\}$$

This makes the computation of $P_x^{(i)}$ easier since $X_1, X_2, \cdots, X_i$ are all binomial random variables. Therefore, the sum of those variables is also binomially distributed with $X_1 + \ldots + X_i \sim B(i \cdot (k+1), p)$ and as a result $P\{X_1 + \cdots + X_i = x + i\}$, i.e., the probability of any combination that sums up to $x + i$, is easy to compute. Note that this latter probability is a good starting point for the computation of $P_x^{(i)}$. However, we need to eliminate from this calculation the probability of any combination of $X_1, X_2, \cdots, X_i$ that results in a decoding in a previous round j. Since a decoding in round $j < i$ requires that $X_1 + \cdots + X_j = j$, this means that $X_{j+1} + \cdots + X_i = x + i - j$. The probability of such a combination happening is clearly $P_i \cdot P\{X_{j+1} + \cdots + X_i = x + i - j\}$ where again $P\{X_{j+1} + \cdots + X_i = x + i - j\}$ is easy to calculate using the Binomial distribution. As a result, $P_x^{(i)}$ can be written as:

$$\begin{aligned} P_x^{(i)} = & P\{X_1 + \ldots + X_i = x\} - \sum_{j=2}^{i-1} P_j \cdot P\{\sum_{l=j+1}^{i} X_l = x + i - j\} \\ & - P\{X_1 = 0\} \cdot P\{X_2 + \ldots + X_i = x + i\} \\ & - P\{X_1 = 1\} \cdot P\{X_2 + \ldots + X_i = x + i - 1\} \end{aligned} \qquad (8)$$

In the previous equation we also take into account the special case where in round 1 there is no lost packet ($X_1 = 0$), so decoding is not required. Note that (8) can be used recursively to compute $P_x^{(i)}$ and therefore $\widetilde{P}_i$ through (5).

Finally, we can compute the average number of packets lost during a decoding failure, $\widetilde{L}$. As we mentioned previously, $W_i = x$ means that $X_1 + X_2 + \ldots + X_i = i + x$,i.e., the total number of lost packets is $i + x$. However, only a percentage of those packets are native. More specifically, it is expected that, on average, $\frac{k}{k+1}(x+i)$ native packets will be lost. Based on this, we can now calculate $\tilde{L}$ as:

$$\widetilde{L} = \sum_{x=1}^{i \cdot k} P\{W_i = x \mid W_j > 0, \ \forall\, j < i\} \cdot \frac{k}{k+1} \cdot (x + i) \qquad (9)$$

Note that, one can use (9) to determine the required coding depth for a specified level of resilience to packet errors. More

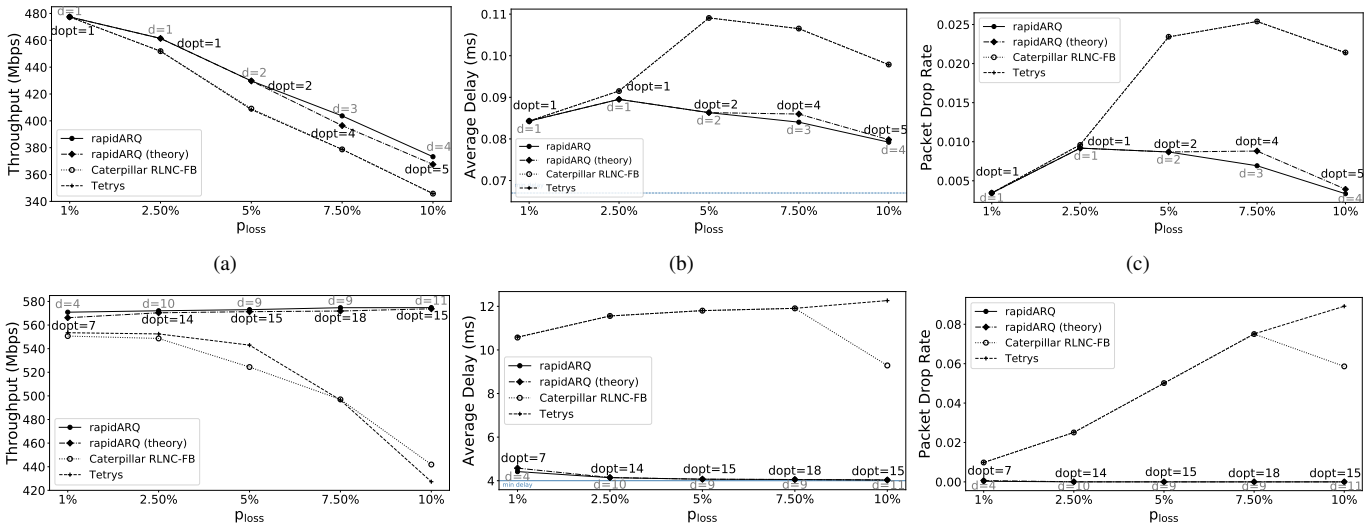


Fig. 5. Performance evaluation of rapidARQ [9], rapidARQ using the analytical prediction of d, Caterpillar RLNC-FB [16] and Tetrys [3] in the 5g ((a)-(c)) and satellite scenarios ((d)-(f)) for different link loss rates (p_l): a),d) throughput performance, b),e) average in-order delivery delay, and c).f) packet drop rate.

specifically, by setting a requirement for $\widetilde{L}$ (e.g., such as 10^{-6} described in URLLC), one can obtain i, i.e., the coding depth. Again, we stress that the values of coding depth obtained through this process are a rough estimation of the optimal coding depth. This is because the optimal value depends on the specific decoding policies implemented at the receiver. However, we will illustrate that the previous analysis can produce a reasonable estimation of the optimal coding depth.

V. Evaluation

We validate the proposed analytical framework through simulation in ns2 simulator [41]. To this end, we use rapidARQ [9], [10] as the protocol that allows manipulating the coding depth. We leverage the analytical framework (see section IV) to obtain the best coding depth value (identified as d_{opt} in the following plots). More specifically, we set a target packet loss rate of 10^{-6} and calculate d_{opt}. We illustrate the version of rapidARQ that uses this value as "rapidARQ(theory)" in the following plots. Furthermore, we present another version of rapidARQ (identified simply as "rapidARQ") where for d we use the smalest value that produces a packet loss rate equal or smaller than 10^{-6}. We identify this value by exhaustively testing a wide range of values. Note that, if such a value does not exist we use the maximum possible value. This can happen when the bandwidth-delay product of the channel only supports small d values (see (3) and the relevant discussion) that can not offer the required level of resilience to errors. The 5G scenario discussed in Section III is such a typical example. We also consider Caterpillar RLNC-FB [16] as a prominent ARQ-based sliding window RLNC protocol, and a Tetrys-like [3] implementation, as the most representative FEC-based sliding window RLNC scheme. We exploit the same evaluation scenarios described in Section III, i.e., a high-speed link to a low-earth orbiting satellite and a typical 5G wireless link. The main simulation parameters are summarized in Table I.

The choice of R is the same for all tested protocols and follows the same principles explained in Section III. That is, R is set according to the link's loss rate (p_l), so that, on average, the introduced redundancy can compensate a single packet loss every $k+1$ packets. As for the channel's error model, we first consider a uniform error model. To ensure that a constant flow of traffic is always available, we use a constant bit rate (CBR) traffic generator. For each experiment, we carry out a set of 5 trials considering a sufficiently large simulation time of $50s$ and report the average values. We assess the performance of the examined protocols based on the throughput, the average per-packet in-order delivery delay and the packet drop rate, i.e., the percentage of source packets that could not be successfully received or decoded.

Figure 5 depicts the protocols' performance for p_l from 1% to 10% in the 5G and Satellite scenarios. For each value of p_l, we illustrate the experimentally identified d value for rapidARQ as well as the one calculated through our analysis (noted as d_{opt}). Clearly, our analysis achieves our goal to produce a good estimation of the best d value that can provide the prescribed level of resilience to losses, i.e. d_{opt} is always close to d. This is extremely important since it eliminates the need of an exhaustive experimental search and allows a more efficient design of such protocols. What is, however, more important is that the d_{opt} values produce a performance very close to the one received when d values, i.e. those identified experimentally. This is true for all three performance metrics, i.e., the throughput, the average delay and the packet drop rate. Moreover, the effectiveness of our analysis is evident for the entire range of p_l and for both tested scenarios although they represent two types of link with very diverse characteristics. Note that, especially for the 5G scenario, there is no d value that can produce a packet drop rate of 10^{-6} or less, as explained previously. However, our analysis can still efficiently approximate the best d value using (3).

As a last note, both variants of rapidARQ massively out-

weigh Caterpillar RLNC-FB and Tetrys protocols. This is due to the appropriate definition of the coding window size according to the channel's loss profile through the use of coding depth. The performance gains are more salient in links with larger bandwidth-delay products, i.e., the satellite link (fig. 5, cases (d)-(f)), since, in the latter, d can be chosen among a wider range of possible values, hence its optimal choice is more critical. While we notice a significant improvement in throughput, the most notable gains pertain to average delay (where the achieved delay in the satellite scenario is three times smaller than in Caterpillar RLNC-FB and Tetrys) and packet drop rate ($\sim 70\%$ reduction in the 5G case). Those results are another confirmation of the importance of correctly setting W (or equivalently d).

VI. Coping with Bursts

Up to now we assumed that a static value of d can provide the desirable protection from channel losses. However, this is true only if the channel's packet loss rate is relatively stable because a static value of d results in a fixed coding performance. If the loss rate varies then there may exist time periods when the introduced redundancy may not be useful because packet losses do not occur. Similarly, there may exist periods of time that the provided protection is not sufficient to cope with bursts of errors. Given the fact that bursty channels are common, it is clear that the coding protocol should be able to realize the emergence of such idle or burst time periods and appropriately adjust its coding depth to reduce/increase its coding efficiency.

In order to achieve the aforementioned functionality, we can leverage the analytical framework proposed in Section IV. Our strategy is to estimate the coding depth value (d_{max}) required for achieving the prescribed loss requirements in the presence of bursts of errors. That is, to assume that losses are uniform and their rate is equal to the one during a burst period. Clearly, this corresponds to the worst-case scenario. Then, we wish to design an adaptive algorithm that can identify the idle or bursty time periods and adjust the coding depth according to the variations of the observed packet error rate. More specifically, under bursts of errors, the adaptive scheme should increase protection, i.e., d, up to the maximum value d_{max} whereas in the opposite scenario, d should be reduced for the sake of complexity gains. *Note that, since our strategy accounts for the worst case scenario (by using d_{max}), its actual target is to benefit by minimizing d during idle periods*. If this minimization is successful, the receiver can experience a small decoding matrix size which directly translates to reduced decoding complexity.

Algorithm 1 illustrates the proposed adaptive algorithm that incarnates our strategy. The estimation d_{max} is derived using the analysis in Section IV for the packet error rate during a burst and is given as input to the algorithm. The adaptive algorithm follows the principle of *fast-increase-slow-decrease*, which signifies an instant reaction in case of estimating an increase in packet losses (lines 1-3) in contrast to a moderate approach in reducing the coding depth when the packet

Algorithm 1 Adapt(lr)

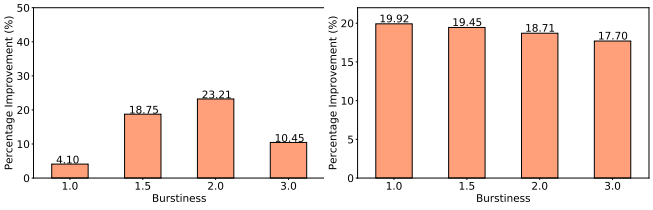
Require: $previouslr, d_{max}, d, k$
1: **if** $lr > previouslr$ **then**
2: $d \leftarrow min\{d_{max}, d+1\}$
3: $W \leftarrow d \times k$
4: **else if** $(lr < previouslr)$ **or** $(lr == 0$ **and** $previouslr == 0)$ **then**
5: **if** successive decrease requests $\geq W/2$ **then**
6: $d \leftarrow max\{1, d-1\}$
7: $W \leftarrow d \times k$
8: remove oldest source packets if current window contents exceed new W
9: **end if**
10: **end if**

TABLE II
Performance of Adaptive vs static rapidARQ for different levels of Burstiness (5G and Satellite scenarios, $RTX=0$).

	LU_{diff} (%)		D_{diff} (ms)		L_{diff}	
B	5G ($d=2$)	Satellite ($d=15$)	5G ($d=2$)	Satellite ($d=15$)	5G ($d=2$)	Satellite ($d=15$)
1	0.19448608	0.00389748	0.0086084	0.0784574	0.0053012	0.0002024
1.5	0.14134774	-0.09113904	0.0093132	1.5515418	0.0060306	0.0032904
2	0.1053732	-0.01337934	0.0095582	1.0156504	0.006386	0.0019448
3	0.12508618	0.08316014	0.0084766	0.0048696	0.005376	-0.0006404

loss rate seems to decline (lines 4-8). The "slow-decrease" approach allows us to avoid dropping out of the coding window unacknowledged packets that are still essential for the decoding process. More specifically, to tackle any unnecessary reduction of d (and equivalently W), we monitor the estimation of packet loss rate. Then, before performing a reduction of d, we require $W/2$ successive observations reporting that this rate either reduces or is zero. We should also bear in mind that if the new value of W is smaller than the number of packets currently in the coding window, we should remove the oldest source packets to ensure consistency (line 8). In order to estimate packet losses, we exploit rapidARQ's cumulative acknowledgements. For stability reasons, the estimation is performed after receiving a prescribed number of ACKs, e.g., equal to the window size $W = d \times k$. Subsequently, the estimated current loss ratio lr is passed as an argument to function *Adapt*, which implements the actual adjustment of d.

We are interested in the comparison of the adaptive scheme, which we refer to as adaptive rapidARQ henceforward, with its static counterpart. We examine performance metrics such as link utilization, delay and loss, but we focus on the complexity signified by the size of the decoding matrix at the receiver. This is because, as explained, the adaptive algorithm, by design, focuses on reducing complexity (through the reduction of d) by taking advantage of the idle periods. Once again, we assess the performance of the examined protocols in both the Satellite and the 5G scenarios without employing re-transmissions ($RTX = 0$). For a fair comparison, we examine the performance of both static and adaptive rapidARQ



(a) (b)

Fig. 6. Percentage improvement of average decoding matrix size vs burstiness of adaptive against static rapidARQ: (a) satellite, and (b) 5G scenario.

TABLE III
PERFORMANCE DIFFERENCE OF ADAPTIVE AND STATIC RAPIDARQ VS PACKET ERROR RATE (SATELLITE SCENARIO, $RTX=0$, $B=2$).

Packet Error Rate p_l	LU_{diff} (%)	D_{diff} (ms)	L_{diff}
2.50%	0.07531078	-0.096753	-0.0005332
5%	0.02326052	0.0129936	0.0002836
7.50%	0.05585636	0.0132738	0.0002524
10%	-0.01337934	1.0156504	0.0019448

considering as coding depth the estimated optimal values for each evaluation scenario, i.e., $d=15$ in the satellite scenario and $d=2$ in 5G. Recall that for the adaptive algorithm those values represent the maximum coding depth. We introduce a bursty error profile using an *on-off* model [10]. In the bursty model, the channel error rate p_l' for the "on" period is set so that during a complete on-off cycle, the error rate is on average equal to 5% for the 5G scenario and 10% for the satellite one. No errors occur during the "off" period. *Burstiness (B)* is expressed as $B = \frac{(\bar{T}_{on}+\bar{T}_{off})}{\bar{T}_{on}}$, where $\bar{T}_{on}$ and $\bar{T}_{off}$ describe the average values of the "on" and "off" period. Therefore, larger B values indicate that $\bar{T}_{on}$ is small and p_l' high.

Table II displays the performance difference between adaptive and static rapidARQ in the satellite and 5G scenarios and for different levels of burstiness. Suppose that, for a specific performance metric M, M_a refers to the metric value for adaptive rapidARQ and M_s for the static protocol. Then, we define the performance difference as $M_{diff} = M_a - M_s$. LU_{diff}, D_{diff} and L_{diff} denote the performance difference in link utilization, delay and loss, respectively. We observe that in both test scenarios adaptive rapidARQ achieves a performance that is comparable to static rapidARQ for all evaluation metrics. This is a clear indication that the adaptive algorithm efficiently identifies bursts and increases d to protect packets. However, when examining the decoding matrix size (Fig. 6), the adaptive scheme achieves significant complexity gains. The plot illustrates the percentage change of the average decoding matrix size that adaptive rapidARQ manages compared to its static counterpart in both the satellite (fig. 6(a)) and in the 5G scenario (fig. 6(b)). At first sight, we observe that adaptive rapidARQ consistently yields improvements. This is a confirmation that the adaptive scheme effectively takes advantage of the idle time periods and reduces the decoding matrix size. In the satellite scenario (fig. 6(a)), the gains for the adaptive scheme increase with burstiness and reach up to an impressive 23%. This is reasonable because increased burstiness translates into longer idle periods. However, a small decline is witnessed for $B=3$, i.e., in the presence of very intense bursts. This is because the adaptive algorithm increases d to a larger extent in order to provide enough protection to tackle any subsequent packet losses. Although the adaptive scheme is able to detect the idle periods, the decrease steps do not suffice to cancel the preceding increments in d, as it happens when B is lower. In the 5G scenario (fig. 6(b)) the gains for the adaptive algorithm are equally impressive (reduction up to $\sim 20\%$). However, the gains depend less on B because the range of d values is very narrow.

Another interesting aspect to explore is how different channel loss rates affect performance assuming fixed burstiness. Here, we only examine the satellite test scenario without retransmissions. We also focus on the case of $B=2$ where the maximum packet loss rate during an "on" period is at least double compared to the average loss rate (p_l). Note that, since we examine various average packet loss rates (p_l), we recalculate the estimation d_{opt} provided by our analytical framework. For the applied link loss rates, the estimated values are: $d_{opt}=14$ for $p_l=2.5\%$, $d_{opt}=15$ for $p_l=5\%$, $d_{opt}=18$ for $p_l=7.5\%$ and $d_{opt}=15$ for $p_l=10\%$. Fig. 7 displays the percentage change of the average decoding matrix size vs p_l. The maximum improvement exceeds 20%. Clearly, there is an increasing trend in gain which follows the increase of the average packet loss rate. In other words, the higher p_l is, the higher the improvements noticed. This is reasonable, since channels with heavy losses require enhanced protection during bursty periods, hence larger fluctuations are noticed in the coding window size. Adaptive rapidARQ is able to identify idle periods and shrink decoding matrix resulting into significant complexity improvements. Interestingly, the observed gains come without sacrificing other aspects of performance. Table III illustrates the performance difference between the adaptive and the static schemes for the link utilization, the average delay and the packet drop rate. In all cases the differences are negligible.

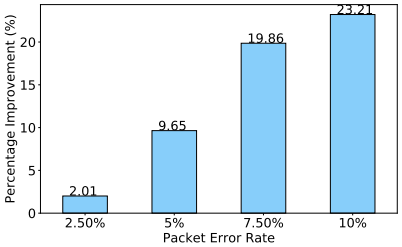


Fig. 7. Average decoding matrix size improvement (%) vs packet error rate for adaptive rapidARQ vs static. Satellite scenario, $RTX=0$, $B=2$.

VII. CONCLUSION

Sliding window RLNC can be a game changer in achieving URLLC's requirements in 5G and beyond networks. However, existing sliding window RLNC schemes tend to neglect the discussion about the importance of an appropriate coding window size, although it plays a major role in the coding scheme's efficiency. In this work, we focused on the impact of the coding window size by utilizing the concept of coding depth d. We experimentally confirmed that the appropriate selection of d is critical in achieving an optimal performance-complexity trade-off. Motivated by this observation, we devised an analytical framework for estimating, based on the channel's loss profile, a suitable d value that produces a near-optimal performance. Furthermore, we explored the dynamic adaptation of d by implementing an efficient adaptive scheme with the aim of minimizing the coding complexity during idle periods. We experimentally confirm both the effectiveness of the analytical model and the adaptive algorithm By transferring such functionality in the transport layer, we anticipate that the benefits will be more intense in multi-hop paths with very

large bandwidth-delay product. Another option would be to allow intermediate nodes to take part in the coding process in order to fully exploit the properties of network coding.

REFERENCES

[1] "Study on scenarios and requirements for next generation access technologies," 3GPP TSG RAN TR38.913 R14, Tech. Rep., 05-2017.

[2] M. Bennis, M. Debbah, and H. V. Poor, "Ultrareliable and low-latency wireless communication: Tail, risk, and scale," *Proceedings of the IEEE*, vol. 106, no. 10, pp. 1834–1853, 2018.

[3] P. U. Tournoux, E. Lochin, J. Lacan, A. Bouabdallah, and V. Roca, "On-the-fly erasure coding for real-time video applications," *IEEE Transactions on Multimedia*, vol. 13, no. 4, pp. 797–812, 2011.

[4] T. T. Thai, J. Lacan, and E. Lochin, "Joint on-the-fly network coding/video quality adaptation for real-time delivery," *Signal Processing: Image Communication*, vol. 29, no. 4, pp. 449–461, 2014.

[5] P. J. Braun, D. Malak, M. Medard, and P. Ekler, "Multi-source coded downloads," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[6] M. Zverev, P. Garrido, R. Agüero, and J. Bilbao, "Systematic network coding with overlap for iot scenarios," in *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2019, pp. 1–6.

[7] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets tcp: Theory and implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, 2011.

[8] M. Zverev, P. Garrido, F. Fernández, J. Bilbao, Ö. Alay, S. Ferlin, A. Brunstrom, and R. Agüero, "Robust quic: Integrating practical coding in a low latency transport protocol," *IEEE Access*, vol. 9, pp. 138 225–138 244, 2021.

[9] F. Karetsi and E. Papapetrou, "A low complexity network-coded arq protocol for ultra-reliable low latency communication," in *2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2021, pp. 11–20.

[10] ——, "Lightweight network-coded arq: An approach for ultra-reliable low latency communication," *Computer Communications*, 2022.

[11] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[12] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," 2003.

[13] N. Papanikos and E. Papapetrou, "Deterministic broadcasting and random linear network coding in mobile ad hoc networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1540–1554, 2017.

[14] J. K. Sundararajan, D. Shah, and M. Médard, "Arq for network coding," in *2008 IEEE International Symposium on Information Theory*. IEEE, 2008, pp. 1651–1655.

[15] J. K. Sundararajan, D. Shah, M. Médard, and P. Sadeghi, "Feedback-based online network coding," *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6628–6649, 2017.

[16] F. Gabriel, S. Wunderlich, S. Pandi, F. H. Fitzek, and M. Reisslein, "Caterpillar rlnc with feedback (crlnc-fb): Reducing delay in selective repeat arq through coding," *IEEE Access*, vol. 6, pp. 44 787–44 802, 2018.

[17] P. Sadeghi, R. Shams, and D. Traskov, "An optimal adaptive network coding scheme for minimizing decoding delay in broadcast erasure channels," *EURASIP Journal on Wireless Communications and Networking*, vol. 2010, pp. 1–14, 2010.

[18] L. Yang, Y. E. Sagduyu, and J. H. Li, "Adaptive network coding for scheduling real-time traffic with hard deadlines," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, 2012, pp. 105–114.

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

[20] 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," *Computer Networks*, vol. 57, no. 6, pp. 1488–1502, 2013.

[21] H.-T. Lin, Y.-Y. Lin, and H.-J. Kang, "Adaptive network coding for broadband wireless access networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 4–18, 2012.

[22] S. Pandi, F. Gabriel, J. A. Cabrera, S. Wunderlich, M. Reisslein, and F. H. Fitzek, "Pace: Redundancy engineering in rlnc for low-latency communication," *IEEE Access*, vol. 5, pp. 20 477–20 493, 2017.

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

[24] T. Van Vu, N. Boukhatem, T. M. T. Nguyen, and G. Pujolle, "Adaptive redundancy control with network coding in multi-hop wireless networks," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 1510–1515.

[25] A. Fu, P. Sadeghi, and M. Médard, "Dynamic rate adaptation for improved throughput and delay in wireless network coded broadcast," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 1715–1728, 2013.

[26] J. Cloud and M. Médard, "Network coding over satcom: Lessons learned," in *International Conference on Wireless and Satellite Systems*. Springer, 2015, pp. 272–285.

[27] P. Garrido, D. J. Leith, and R. Agüero, "Joint scheduling and coding for low in-order delivery delay over lossy paths with delayed feedback," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1987–2000, 2019.

[28] A. Cohen, D. Malak, V. B. Bracha, and M. Médard, "Adaptive causal network coding with feedback," *IEEE Transactions on Communications*, vol. 68, no. 7, pp. 4325–4341, 2020.

[29] A. Cohen, G. Thiran, V. B. Bracha, and M. Médard, "Adaptive causal network coding with feedback for multipath multi-hop communications," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 766–785, 2020.

[30] M. Karzand, D. J. Leith, J. Cloud, and M. Medard, "Design of fec for low delay in 5g," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 8, pp. 1783–1793, 2017.

[31] J. Cloud and M. Médard, "Multi-path low delay network codes," in *IEEE Global Communications Conference*. IEEE, 2016, pp. 1–7.

[32] M. Karzand and D. J. Leith, "Low delay random linear coding over a stream," in *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2014, pp. 521–528.

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

[34] S. Wunderlich, F. Gabriel, S. Pandi, F. H. Fitzek, and M. Reisslein, "Caterpillar rlnc (crlnc): A practical finite sliding window rlnc approach," *IEEE Access*, vol. 5, pp. 20 183–20 197, 2017.

[35] M. Brulatout, H. Khalifé, V. Conan, J. Leguay, E. Lochin, and J. Lacan, "Reliable streaming protocol for lossy networks," in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2015, pp. 1486–1491.

[36] E. Tasdemir, M. Tömösközi, J. A. Cabrera, F. Gabriel, D. You, F. H. Fitzek, and M. Reisslein, "Sparec: Sparse systematic rlnc recoding in multi-hop networks," *IEEE Access*, vol. 9, pp. 168 567–168 586, 2021.

[37] P. Garrido, D. Gómez, J. Lanza, and R. Agüero, "Exploiting sparse coding: A sliding window enhancement of a random linear network coding scheme," in *IEEE International Conference on Communications*, 2016, pp. 1–6.

[38] T. Geithner, F. Sivrikaya, and S. Albayrak, "Joint link rate selection and adaptive forward error correction for high-rate wireless multicast," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 830–846, 2022.

[39] V. Roca and B. Teibi, "Sliding window random linear code (rlc) forward erasure correction (fec) schemes for fecframe," Internet Requests for Comments, RFC Editor, RFC 8681, 01 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8681

[40] J. Heide, M. V. Pedersen, F. H. Fitzek, and T. Larsen, "Network coding for mobile devices-systematic binary random rateless codes," in *IEEE International Conference on Communications Workshops*, 2009, pp. 1–6.

[41] S. McCanne and S. Floyd. ns Network Simulator. [Online]. Available: http://www.isi.edu/nsnam/ns/

[42] ns-3 Network Simulator. [Online]. Available: https://www.nsnam.org/

[43] 3GPP, "Study on non-orthogonal multiple access (noma) for nr," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.812, 12 2018, version 16.0.0.

[44] ——, "Study on using Satellite Access in 5G:Stage 1 (Release 16)," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.822, 06 2018, version 16.0.0.