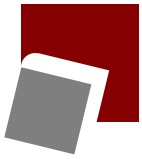


Institute for Advanced Studies
in Basic Sciences
Gava Zang, Zanjan, Iran

Institute for Advanced Studies in Basic Sciences

Department of Computer Science and Information Technology

Partial Packet Recovery for Network Coding

Master of Science

in

Computer Science (Artificial Intelligence)

Hosein K. Nazari

Supervisor:

Dr. Peyman Pahlevani

August, 2021

Abstract

Emerging applications demand more reliability and lower latency than ever before. However, improving reliability, particularly for noisy wireless communication, remains a challenging task. During transmission of packets over a noisy communication channel, some of them might partially corrupt. These packets are referred to as partial packets.

In early 2000, network coding has emerged as a compelling solution to improve resource utilization and reliability. Originally, network coding discards partial packets, which is not an efficient strategy. The main reason is that a significant portion of partial packets contains error-free and valuable information. Several partial packet recovery solutions have been suggested as complementary strategies to exploit partial packets. However, they suffer from high computational complexity and long decoding delay.

In this study, we propose two partial packet recovery techniques for noisy wireless communications. First, we introduce F-PRNC, a partial packet recovery technique that focuses on enhancing packet recovery speed. Alongside the complexity analysis, we carried out extensive simulations and compared F-PRNC with state-of-the-art solutions. In severely noisy situations, the simulation findings show that F-PRNC can boost goodput more than four times and minimize decoding delay by 4.5 times. Also, we have proposed Fly-PRAC with a focus on reducing the estimation and decoding delay. This method is the first partial packet recovery technique that enables the recovery process at intermediate nodes. Simulation shows that Fly-PRAC reduces completion time by 16 percent and increases goodput up to 3.8 times in highly noisy communication channels. It is also helpful for sparse network coding based communications, as it reduces average decoding delay by 47 percent, making it more suitable for a range of real-time applications.

Keywords: partial packet recovery, random linear network coding, network coding.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we state the importance of reliable communication in a noisy wireless channel. Then, we briefly introduce network coding and packet recovery techniques to combat issues caused by noise. Finally, we outline the motivation of this study and present our contribution.

1.1 Reliable comunication in a noisy wireless channel

Wireless communication has been expanding at an astonishing rate over the last few decades. This type of communication provides more flexibility and has a broader range of applications compared to wired communications.

There are different categories and applications of wireless communications. For instance, short-range communications such as Bluetooth [2], ZigBee[3], and Wi-Fi provide different data rates and are used in machine-to-machine (M2M) communications or to build personal area networks [4]. The medium-range cellular network such as 4G/LTE and 5G are used to provide internet access for roughly 4 billion devices across the world [5]. Also, low-power wide-area networks (LPWAN) are

using wireless transmitting techniques such as Sigfox, LoRa, and NB-IoT to connect millions of internet of thing (IoT) devices [6].

These vast number of applications of wireless communication have different and sometimes conflicting requirements. One of the most wanted requirements is to provide reliable communication. A communication channel is reliable if it delivers a set of packets ordered, without any losses or duplicates [7]. Each channel, regarding its characteristics, has a certain capacity. The capacity of a channel indicates a tight upper bound on the maximum rate at which data is transmitted reliably.

Due to the faulty nature of wireless communication channels, packets can corrupt during transmission. There are several reasons for this, such as signal authentication, signal congestion, or multi-path propagation [8]. These problems can degrade the capacity and reduce the performance of a communication channel.

Maintaining reliable communication has always been a hot spot in the field of computer networks for a long period, and it remains a challenging task. Conventionally, techniques used to increase reliability for a communication channel falls into two main categories: active retransmission and passive channel coding. In active retransmission methods such as automatic repeat request (ARQ), the receiver discards corrupted packets and issues a retransmission request. In passive channel coding, also known as the forward error correction (FEC), the encoder adds redundant information to each packet and then use that to recover corrupted part of the packets. The ARQ is not suitable for very noisy environments or broadcast scenarios, and FEC has limited correction capabilities and only performs well within a specific range of the Signal-to-noise ratio(SNR) [9] [10].

1.2 Network Coding

In [11], Ahlswede et al. propose a novel concept called Network Coding (NC) that has arisen as a possible solution to resource utilization and gaining higher reliability [12][13][14]. NC is also beneficial for storage area networks, peer-to-peer file sharing, multicast streaming, and wireless mesh networks [15][16].

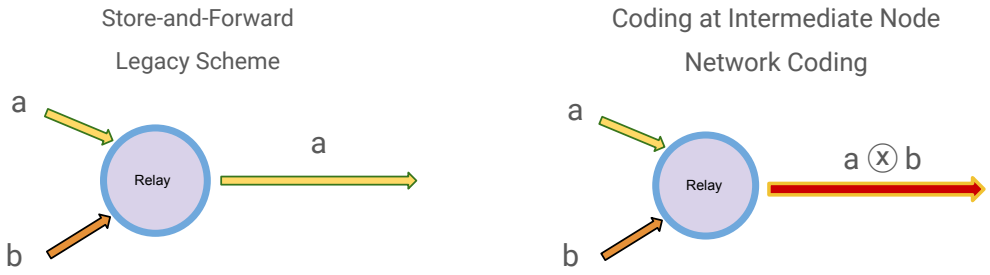


Figure 1.1: Comparison of network coding and store-and-forward scheme; network coding allows intermediate nodes to recode packets without decoding them first.

Figure 1.1 shows this networking technique allows the intermediate nodes to mitigate the value of packets. In the old-fashioned communications, i.e., store and forward method, the received packets are cached and then transmitted to the next node.

1.2.1 Linear Network Coding

Here [17], Ho et al. introduced RLNC, a novel random strategy for decentralized NC. In RLNC, each data block is divided into a group of g source packets called generation. On the sender side, n coded packets are generated by linearly combining source packets where $n \geq g$. Once g linearly independent coded packets are received, the decoding process begins on the receiver side.

Using RLNC increases the loss-resilience of communication; however, its major drawback is the

high complexity of the decoding process, which is $O(g^3)$. Also, it starts decoding only after receiving g linearly independent packets causing a considerable decoding latency. These are two main barriers of this approach [18][19]. SNC has been introduced to address these issues and reduce the decoding complexity by using fewer source packets to form a coded packet [19] [20]. It is best suited for real-time applications requiring a low decoding delay. However, using fewer source packets in forming coded packets result in a dramatic increase in transmission overhead [21][22].

1.2.2 Packet Recovery for Linear Network Coding

In NC, the integrity of received packets is checked using an error detection code such as a cyclic redundancy check (CRC). If any packet contains corrupted information, the CRC refutes its integrity. Here we refer to it as a partial packet.

Originally in RLNC and SNC, partial packets are discarded. In [23], it is shown roughly 55% of packets in IEEE 802.11a/b/g are partial. Also, in [24], authors demonstrate that up to 95% of a partial packet consists of useful and error-free information. Therefore, discarding them results in poor bandwidth utilization. In this regard, to achieve higher reliability and throughput, partial packet recovery. (PPR) techniques are employed as a complementary strategy.

There are several PPR methods for NC communications. These methods aim to recover partial packets and avoid more re-transmission. Also, these methods generally consist of two steps, namely, estimation and correction. In the estimation step, the majority of PPR techniques for NC, such as [25] [1], use a method called algebraic consistency rule (ACR) checks. ACR benefits from the algebraic relation between coded packets in NC and estimates error locations for received packets. The PPR uses the reported locations and attempts to recover them during the correction step.

For this purpose, it modifies the information in the reported locations multiple times. After each modification, it checks the integrity of information in the reported locations using a checking mechanism. This checking mechanism is either ACR or CRC check. The process of modification and checking continues until the checking mechanism verifies the integrity of the information.

1.3 Problem statement

PPR methods for NC can help to reduce the number of required re-transmissions and take advantage of partial packets. However, these methods are only considered point-to-point communications where only one sender and a receiver communicate directly. Therefore, in more realistic scenarios with intermediate nodes, only the receiver can recover packets. As previously stated, one of the primary advantages of NC over other coding systems is the ability to recode packets at the intermediate node. In more realistic settings with numerous relays, partial packets cannot be used in the recoding process, which negatively influences the entire process and reduces throughput.

Furthermore, the estimation and correction process only begins after receiving $g + 1$ packets causing a significant delay in the decoding process. Also, the correction process is responsible for a significant portion of time spent on the recovery process, which grows dramatically when the error rate, the number of source packets, or the size of the packets increases. As a result, they are impractical in a range of situations.

Additionally, there is a risk of false-positive events in the estimation and correction processes, where they may incorrectly verify an erroneous block of information or cause a failure in the recovery process.

1.4 Contribution

The main contribution of the present study is proposing two PPR methods for NC communication, namely fast packet recovery for network coding (F-PRNC) and Fly-PRAC.

The F-PRNC [26] aims to reduce the recovery time and increase goodput in highly noisy environments. F-PRNC appends redundancy symbols to each coded packet, which allows it to begin the recovery process only after receiving the first partial packets. When a sufficient number of packets arrive, F-PRNC employs ACR to estimate error locations and attempts to recover the remaining unrecovered packets. Alongside the analytical complexity analysis of the decoding process, we have carried out extensive simulation and compared the F-PRNC with the two latest PPR schemes for NC, DAPRAC [1] and S-PRAC [27]. We show that F-PRNC exceeds those methods in terms of goodput in a variety of settings. Also, simulation reveals that F-PRNC for highly noisy channels with byte error rate 0.14 has dramatically lower retransmission and decoding delay.

Fly-PRAC is another PPR scheme that considers more realistic communications. This method allows intermediate nodes to recover partial packets and use recovered ones in the re-coding process. This feature permits communication to utilize NC benefits fully. In Fly-PRAC, the transmitter sends several intentionally linearly dependent packets and uses them in the estimation and correction process. Fly-PRAC, in addition to RLNC, performs well in SNC. It takes advantage of redundant packets in SNC to recover partial packets, reduce total transmissions, and decrease average decoding delay.

We also propose a new estimation technique based on algebraic consistency among packets. Unlike ACR, which requires at least $g + 1$ packets to begin estimating, the proposed approach

requires a smaller set of R packets. This technique allows us to begin estimation and correction steps much earlier and reduce decoding delay. It also helps to provide a more precise estimation of error locations which boosts the correction phase. We also demonstrated proposed method lowers the probability of false-positive events. As a result, Fly-PRAC recovers more partial packets and reduces transmission overhead.

We have conducted extensive simulations on Fly-PRAC's performance and compared it with the S-PRAC in various conditions. The results illustrate that Fly-PRAC dramatically enhances communication efficiency in poor channel conditions. For instance, in a point-to-point RLNC communication, for a bit error rate of 5×10^{-5}, the completion time can be decreased up to 4.7 times. Also, it illustrates that for large payload sizes such as 900 bytes, the proposed method can increase the goodput up to 3.8 times. Further, for a large number of source packets such as 200 bytes, results demonstrate that it recovers more packets and requires 21% fewer packets to decode all source packets. Additionally, in an SNC communication and a bit error rate of 10^{-4}, results show employing Fly-PRAC leads to a reduction of up to 47% in average decoding delay. Lastly, it is shown that enabling packet recovery at intermediate nodes boost the performance, and for a channel with a bit error rate of 10^{-4}, this can reduce the total transmission up to 16%.

* As an outcome of this study, we published a paper entitled "Improving the Decoding Speed of Packet Recovery in Network Coding" in the IEEE Communications Letters journal. We also submitted a paper entitled "Fly-PRAC: Partial Packet Recovery for Linear Coded Transmission" for review.

1.5 Thesis Structure

The remainder of this thesis is structured as follows. The second chapter goes through the basics of network coding, random linear network coding, and sparse network coding. We also briefly review some background information about noises in wireless communications. The third chapter presents two partial packet recovery techniques for network coding and describes their encoding, recovery, and decoding procedures. In chapter 4, we have the analysis section. Then, we report the outcome obtained by simulation results and compare proposed methods with the most relevant PPR schemes in the fifth chapter. Finally, in chapter 6, we first conclude this study and then discuss potential improvements that will be addressed in future research.

Chapter 2

Background and Related Work

2.1 Introduction

In this chapter, we first review some background information about NC and its different implementations. Following that, we go through the RLNC in-depth, explaining its encoding and decoding procedures. Afterward, we discuss SNC and highlight its main differences with RLNC. Further, we discuss noises in communication channels, their different types, and mathematical models. In the related work section, we briefly discuss traditional methods of coping with errors in communication channels. Then we look at different types of PPR before looking deeper into PPR schemes for NC communications and discussing their benefits and drawbacks.

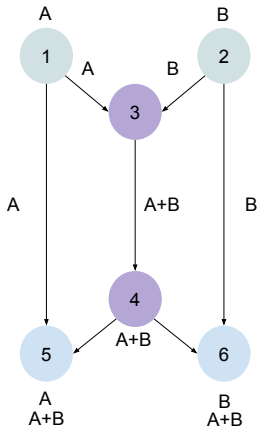


Figure 2.1: Network coding helps to increase the throughput gain in the "Butterfly Network"

2.2 Background

2.2.1 Network Coding Implementations

Network coding [11] is a unique concept that has emerged as a potential solution to resource utilization and higher reliability. The NC concept entails combining and re-transmitting a combination of packets at the network's source and intermediate nodes. This type of communication has proven to be more resilient to packet losses and more adaptable to network changes [15][16].

Figure 2.1 depicts the butterfly network. It shows a network of six nodes, nodes 1 and 2 serving as a source, nodes 3 and 4 serving as a relay, and nodes 5 and 6 are sink nodes. Furthermore, each edge represents a communication channel that transmits one packet in a time slot. Each source node transmits a message, and two sink nodes must receive both messages. Once node 3 receives both messages A and B in a time slot, instead of forwarding them in two time-slots, encodes them by adding them together and retransmits the resulting packet to the downstream node. No routing

technique could achieve such efficiency if the relay node 3 could carry either A or B. The sink nodes 5 and 6 directly receive A and B from side edges. Next, by receiving the encoded packet A+B from the intermediate node 4, they can obtain the other message with a simple operation. It can be observed that NC greatly improves the throughput and power efficiency due to the fewer time slots needed for transmitting the same amount of information.

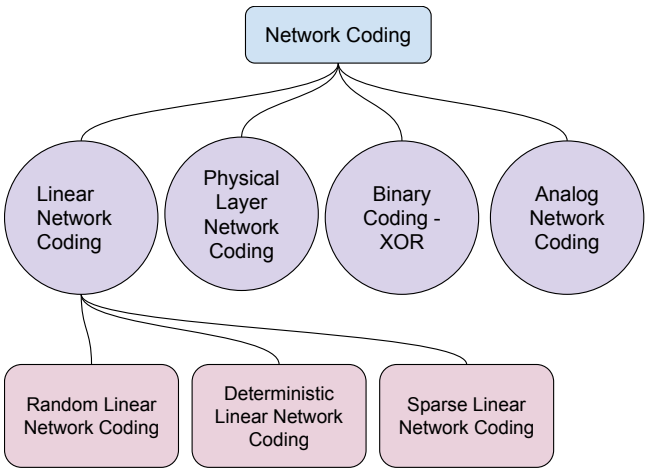


Figure 2.2: Categorization of network coding implementations.

There are numerous methods for performing NC depending on the operation used to combine packets and the network layer in which this operation occurs. In [28], they divided them into four different categories namely physical layer network coding [29], analog network coding [30], binary coding-XOR [31], and linear network coding. The linear network coding also contains three subcategories. These categories are random linear network coding (RLNC)[32], sparse linear network coding (SNC)[19], and deterministic linear network coding [33].

2.2.1.1 RLNC

Here [17], Ho et al. introduced RLNC, a novel random strategy for decentralized network coding. RLNC divides data block into g source packets $\{p_1, p_2, ..., p_g\}$ called a generation. Each p_i contains l symbols $\{s_1, s_2, ..., s_l\}$. All symbols are elements of a finite field $GF(2^q)$.

For each coded packet, there is an associated coefficient vector. Let $c_i = \{c_{i,1}, c_{i,2}, ..., c_{i,g}\}$ denoted a coefficient vector that is associated with the i-th coded packets (p'_i). All elements of coefficient vectors are also selected randomly and independently from a finite field $GF(2^q)$. The encoder constitutes two matrices; one containing packet and another containing coefficient vectors and uses Eq. 2.1 to form coded packets.

$$\begin{pmatrix} p'_1 \\ p'_2 \\ \vdots \\ p'_n \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,g} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,g} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,g} \end{pmatrix} \times \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,l} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ s_{g,1} & s_{g,2} & \cdots & s_{g,l} \end{pmatrix}. \tag{2.1}$$

Each coded packet contains g coded symbols. Let $s'_{i,j}$ refer to the j-th coded symbol of i-th coded packets. The receiving node needs to receive at least g linearly independent coded packets. Once a sufficient number of packets are received, the decoding process begins. Received packets and their associated coefficient vectors are used to form a matrix denoted by M. The matrix M also can be seen as a set of equations, with g unknowns.

$$M = \left(\begin{array}{cccc|cccc} c_{1,1} & c_{1,2} & \cdots & c_{1,g} & s'_{1,1} & s'_{1,2} & \cdots & s'_{1,g} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,g} & s'_{2,1} & s'_{2,2} & \cdots & s'_{2,g} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,g} & s'_{n,1} & s'_{n,2} & \cdots & s'_{n,g} \end{array}\right), \tag{2.2}$$

where n shows the number of received packets. The decoding algorithm is essentially a step-by-step Gaussian elimination [34] over the finite field. The Gaussian elimination algorithm performs over matrix M to obtain the row echelon form and retrieve source packets. This process has a computational complexity of $O(g^3)$.

$$M = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & s_{1,1} & s_{1,2} & \cdots & s_{1,g} \\ 0 & 1 & \cdots & 0 & s_{2,1} & s_{2,2} & \cdots & s_{2,g} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & s_{n,1} & s_{n,2} & \cdots & s_{n,g} \end{array}\right), \tag{2.3}$$

2.2.1.2 SNC

SNC has been introduced to address the high computational complexity of the decoding process in RLNC. In the encoding process, the encoder selects the coefficient vectors sparsely. In SNC, each coded packet associates with a coding vector that contains only w non-zero elements that are selected randomly and independently from a finite field $GF(2^q)$. SNC is best suited for real-time applications requiring a low decoding delay. However, the major disadvantage of SNC is its transmission overhead. Since the w non-zero elements are selected randomly, and the coefficient vector is sparse, fewer source

packets are combined in a coded packet. As a result, some of the received coded packets may contain information from source packets that have already been decoded on the decoder side. As we decrease w, the transmission overhead gets more dramatic [35].

2.2.2 Modeling a Noisy Channel

Due to the faulty nature of communication channels, packets can corrupt during transmission. This is more common with wireless communications, and there are several reasons for it, such as signal authentication, signal congestion, or multi-path propagation. The main types of errors are single-bit and burst errors. In single-bit errors, only one bit changes for a given data unit. This type of error is least likely since the duration of noise is generally longer than the duration of transferring only one bit.

On the other hand, burst errors generally are more likely to happen, and it refers to more than one-bit corruption for a given data unit such as a packet, a byte, and a character.

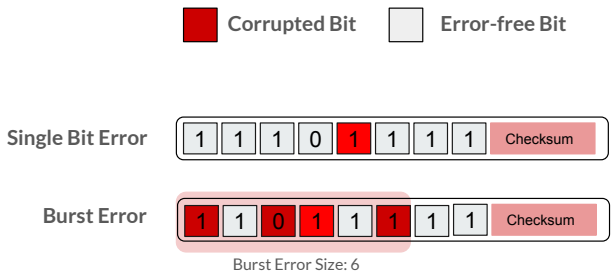


Figure 2.3: Examples of single-bit and burst error in a packet.

Noisy channels can be modeled in several ways. Here we mention two of them, binary symmetric channel (BSC) and binary erasure channel (BEC). In the case of sending only one bit in BCS, the bit is preserved with the probability of $1 - p$, and there is a chance of negation with the probability

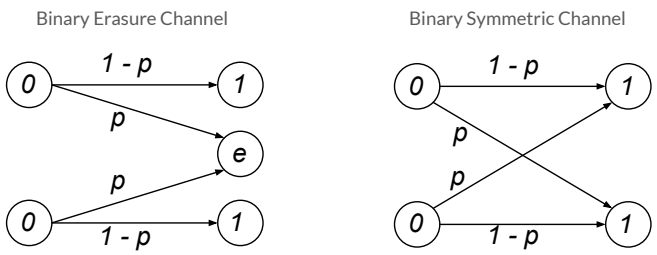


Figure 2.4: Binary erasure and binary symmetric channels.

of p. In BEC, the channel either preserves input with the probability of $1-p$ or erases it with the probability of p. In this study, we consider the former model as a base for our study.

2.3 Related works

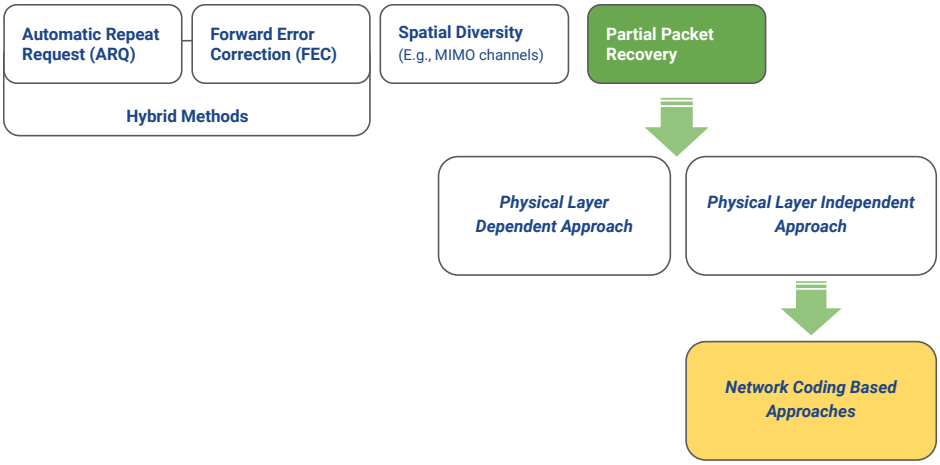


Figure 2.5: Categorization of reliability methods.

2.3.1 FEC and ARQ

Traditionally, FEC and ARQ are widely being used to combat the erroneous channel's challenges and provide higher reliability. FEC appends a constant overhead to packets and provides limited

correction capabilities. Due to fast-varying and unstable channel quality in wireless networks, it is necessary to consider the worst-case scenario to achieve high reliability. This eventually leads to inefficient utilization of network resources. Moreover, acknowledgment packets can provide reliability in some conditions, although it seems impractical in poor channel conditions and broadcast scenarios.

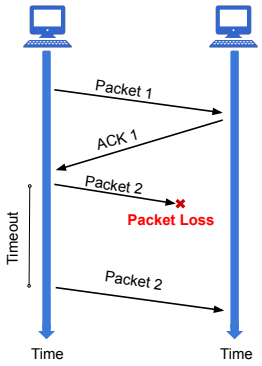


Figure 2.6: An example of automatic repeat request method.

Hybrid-ARQ exploits both FEC and ARQ to propose a more flexible method that performs better than ARQ. In Hybrid-ARQ, before transmission of each packet, FEC and error-detecting codes are appended to reduce the number of needed re-transmissions. However, in good channel conditions, the overhead relating to the appended data degrades the throughput.

2.3.2 Spatial Diversity

Spatial diversity suggests dedicating more than one antenna to each node in a wireless network. This way, nodes can send or receive parallel streams of redundant information and use this different version of received data in the correction process. For instance, multiple-input multiple-output (MIMO) is an implementation of spatial diversity. In some cases, one receiver may employ two antennas for receiving a parallel stream of information from different paths. It compares the strength of received

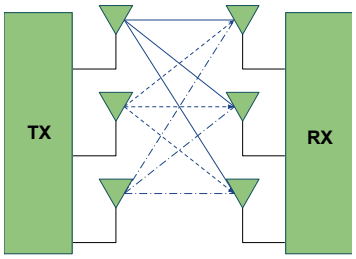


Figure 2.7: An example of multi-input multi-output transmissions using 6 antennas.

signals and chooses the stronger one. As it is unlikely that both paths degrade simultaneously, this technique helps achieve higher reliability at the expense of more hardware requirements and receiving unnecessary, redundant information in good channel conditions.

2.3.3 Partial Packet Recovery

Partial packet recovery (PPR) was first proposed in [36]. The main challenge of any PPR approach is the detection of error locations. A new architecture called SOFT for the physical layer is introduced in [37], which conveys a measure of confidence for each received bit. In [38], a link-layer retransmission protocol called PP-ARQ is introduced. In PP-ARQ, the receiver can determine which part of the packet is received correctly using acknowledgments and only requests the retransmission of the corrupted parts. In segment-based PPR approaches [39][40], the packet is divided into segments, and the receiver requests the retransmission just for the corrupted segments.

A range of techniques utilizes cross-layer information. For instance, approaches based on soft-decoding use a confidence measure provided by the physical layer to estimate the location of erroneous bits. These measurements are sent to higher layers, where the receiver issues a request to re-transmit those bits. Besides its advantages, since passing such information violates the current layered network protocols and standards, it also requires modification at the hardware level [41].

Moreover, a variety of packet recovery methods are proposed specifically for NC. In figure 2.8, we have classified them into two different groups based on techniques used to detect corrupted parts of a partial packet. Both groups rely on linear relations between coded packets to estimate corrupted parts of a packet and do not require hardware modification. In the following, we briefly review them.

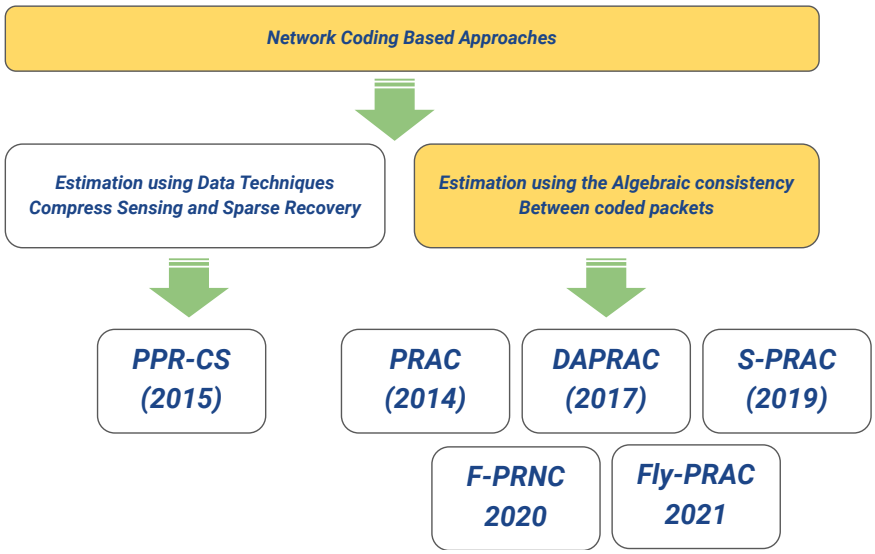


Figure 2.8: Classification of partial packet recovery methods for network coding communications.

2.3.3.1 PPR-CS

A variety of packet recovery techniques are proposed specifically for NC. PPR-CS [42][43] relies on data processing techniques and works transparently from the sender. PPR-CS uses a systematic RLC encoder and decoder, exploits the coding matrix's algebraic properties, and uses compress sensing to recover partial packets. However, It relies on the physical layer to correct most bit errors using FEC techniques and assumes that errors are sparse, random, and independently distributed among different packets. For poor channel conditions, the smaller generation sizes are suggested to

decrease the sparsity order of errors.

2.3.3.2 PRAC

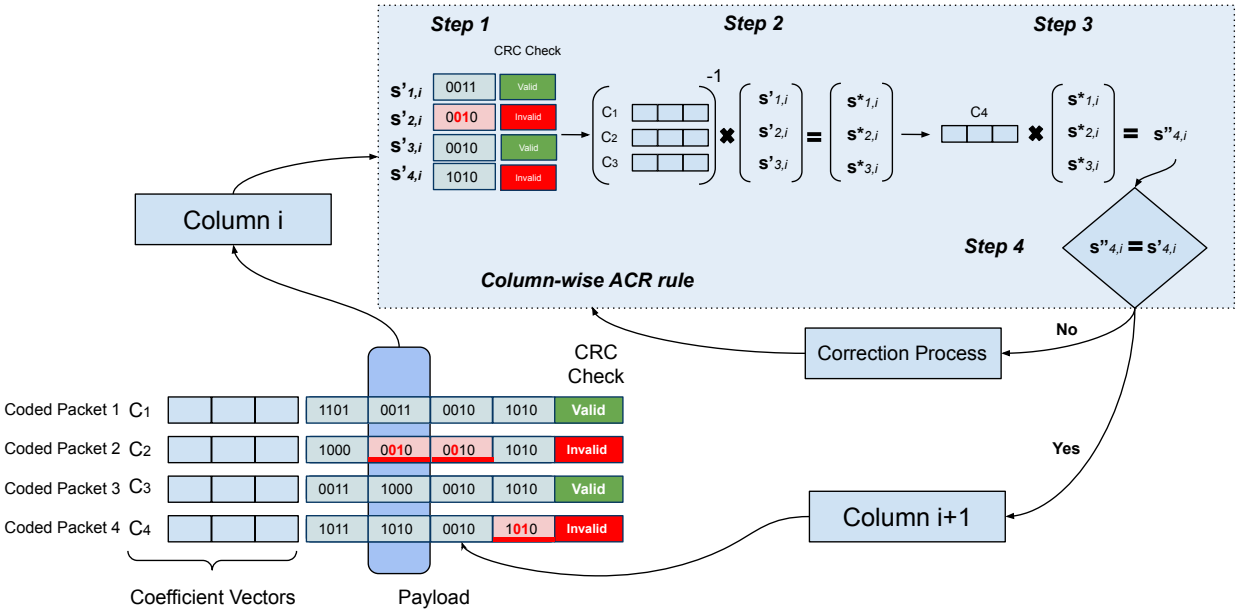


Figure 2.9: Estimation and correction process of PRAC.

Packetized rateless algebraic consistency (PRAC) [25], is the first approach that suggests exploiting the algebraic consistency of coded packets to estimate the location of erroneous symbols. It begins the recovery process after collecting more than g packets. Figure 2.9 shows ACR steps for a generation of three source packets. In this example, to begin the ACR process, four coded packets are required. The ACR check performs column by column and aims to check the integrity of a set of coded symbols present in that column. To check the integrity of the i-th column, it first selects coded symbols of coded packets 1 to 3. In the second step, the selected coded symbols are used to obtain estimated symbols. In the third step, the estimated symbols are used to re-code the coded symbol of the fourth

packet ($s''_{4,i}$). If $s''_{4,i}$ is equal to the received version of coded symbols ($s'_{4,i}$), the ACR check verifies the integrity of that column. Columns with verified ACR checks are referred to as the consistent column, and if the ACR check refutes a column's integrity, we refer to it as an inconsistent column. PRAC tries to recover inconsistent columns, and for this purpose, it permutes the value of symbols involved in that columns. After each permutation, PRAC uses ACR to check if the permutated version of the column is consistent or not. These permutations and checks are continued until the ACR check verifies the column.

The ACR check, may incorrectly verify the integrity of corrupted information. Assume a generation with two source packets containing only one segment. Let $s_{1,1} = 1$ and $s_{2,1} = 3$ be the symbols of source packets and $C_1 = \{c_{1,1} = 1, c_{1,2} = 2\},\ C_2 = \{c_{2,1} = 1, c_{2,2} = 2\},\ C_3 = \{c_{3,1} = 1, c_{3,2} = 2\}$ be the coefficient vectors of three coded packets.

$$\begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \\ c_{3,1} & c_{3,2} \end{bmatrix} \times \begin{bmatrix} s_{1,1} \\ s_{2,1} \end{bmatrix} = \begin{bmatrix} 7 \\ 5 \\ 4 \end{bmatrix}. \tag{2.4}$$

According to Eq. 2.4, $s'_{1,1} = 7,\ s'_{2,1} = 5,\ s'_{3,1} = 4$ are the coded symbols. Assume first two received packets has been corrupted during the transmission and the received coded symbols are $s'_{1,1} = 6,\ s'_{2,1} = 6$ and $s'_{3,1} = 4$. To check the integrity of received symbols, we use first and second received packets to estimate source packets.

$$\begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}^{-1} \times \begin{bmatrix} s'_{1,1} \\ s'_{2,1} \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}. \tag{2.5}$$

According to Eq.2.5, the estimated source packets are $s^*_{1,1} = 2,\ s^*_{2,1} = 2$. In the next step of ACR, estimated source packets are used to re-code the third packet.

$$\begin{bmatrix} c_{3,1} & c_{3,2} \end{bmatrix}^{-1} \times \begin{bmatrix} s^*_{1,1} \\ s^*_{2,1} \end{bmatrix} = 4. \tag{2.6}$$

According to Eq.2.6, the re-coded version of third packets($s''_{1,3}$) is equal to 4 and $s''_{1,3} = s'_{1,3} = 4$. Therefore, the ACR incorrectly verifies the integrity of received symbols while the estimated version of symbols ($s^*_{1,1}$ and $s^*_{2,1}$) are not equal to symbols ($s_{1,1}$ and $s_{2,1}$). In the rest of present work, we refer to this as a false-positive event in ACR check.

PRAC does not need any overhead over packets and boosts the throughput in good channel condition. Still, in poor conditions with higher error rates, PRAC suffers from a slow recovery process making it impractical for larger generation and packet sizes. Another drawback of PRAC is that the recovery process begins after receiving $(g + 1)$-th packets causing a huge decoding delay.

2.3.3.3 DAPRAC

In [1], DAPRAC proposed to address the poor speed of packet recovery. It uses different estimation and correction procedures to reduce the search space.

In DAPRAC, a CRC denoted by inner CRC is calculated and appended to each source packet before encoding packets. Upon encoding packets, another CRC called outer CRC is calculated and appended to each coded packet. DAPRAC uses these CRCs in the estimation and correction phase.

Figure 2.10 and 2.11 are an example of estimation and correction process of DAPRAC. In this example, DAPARAC recovers partial packets for a generation of 2 source packets. On the receiver

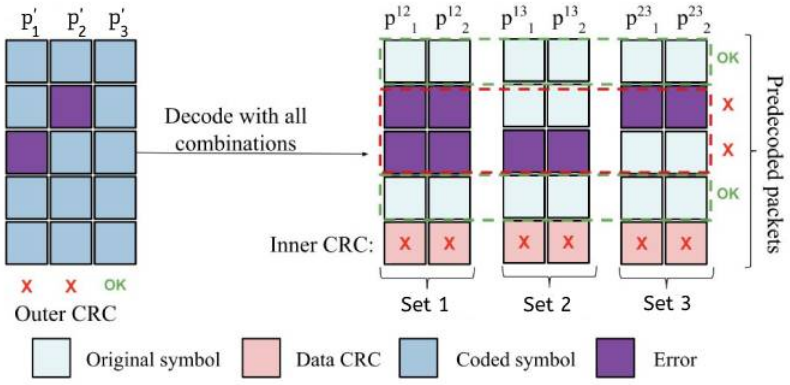


Figure 2.10: Error estimation process of DAPRAC. Image from [1].

side, three packets have arrived. First, DAPRAC tries to decode packets using every combination of 2 coded packets out of 3 received ones. It refers to decoded packets as pre-decoded packets. Figure 2.10 shows three sets of pre-decoded packets where $p_k^{i\ j}$ refers to pre-decoded packet k decoded using received coded packets p'_i and p'_j. DAPRAC detects the location of errors by finding differences between pre-decoded packets when comparing symbol-wisely. Figure 2.11 shows the correction process of DAPRAC. First, it selects one set of pre-decoded packets. Next, to recover pre-decoded packets with invalid inner CRC for all detected error locations, DAPRAC changes the symbols of chosen packets with symbols of the exact location of other pre-decoded packets. After each change, the integrity of pre-decoded packets is checked with inner CRC and continues changing and checking until the inner CRC verifies the pre-decoded packets. These pre-decoded packets with verified inner CRCs are the recovered versions of source packets.

In case of any errors in the same part of packets or inner CRCs, the recovery process fails. Besides, providing pre-decoded packets is time-consuming and causes delay, especially for a high number of source packets. Further, similar to PRAC, it can begin the recovery process only after

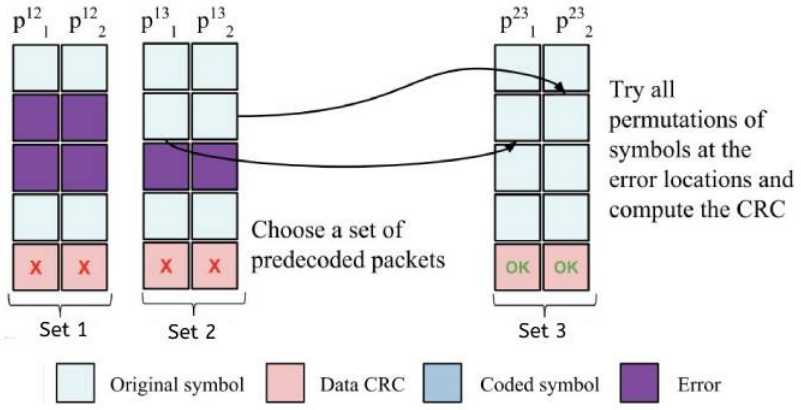


Figure 2.11: Correction process of DAPRAC. Image from [1].

receiving at least g coded packets.

2.3.3.4 S-PRAC

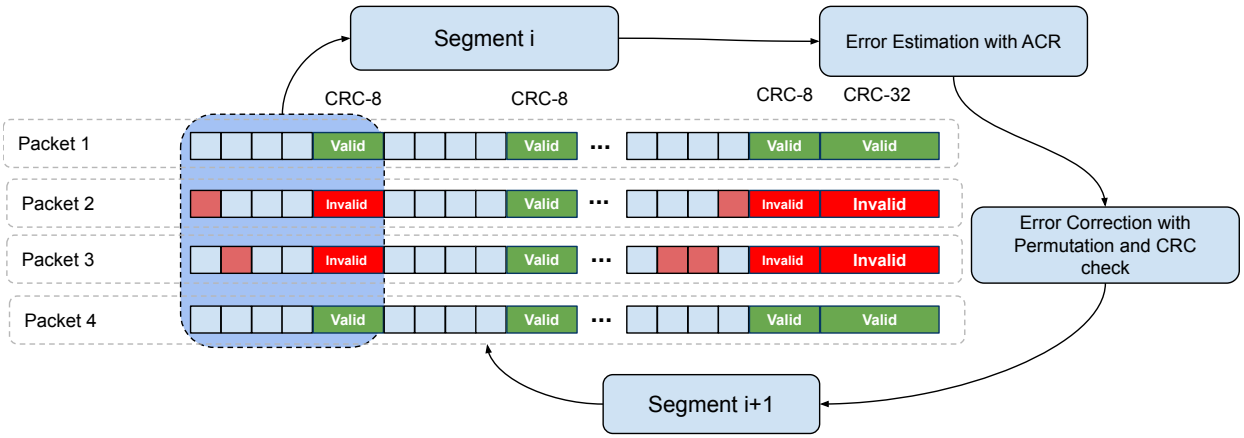


Figure 2.12: Estimation and correction process of S-PRAC.

To address the same issue, authors of S-PRAC [27], suggest splitting each coded packet into s segments and computing one CRC denoted by ICRC for each one. It also considers a CRC computed for all segments called OCRC and assumes that the CRCs are immune to error. After collecting g

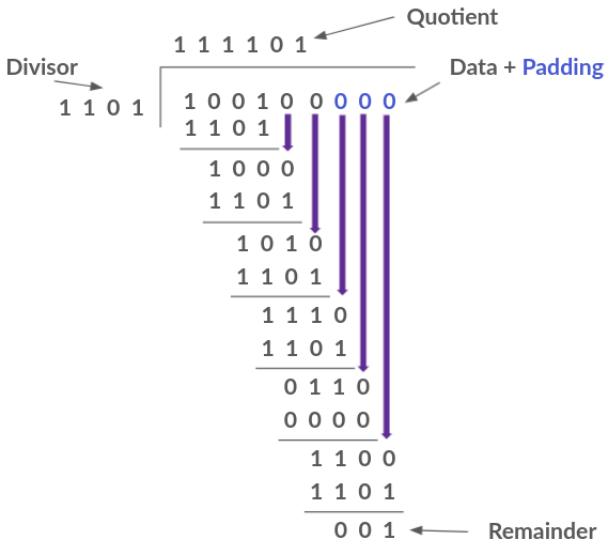
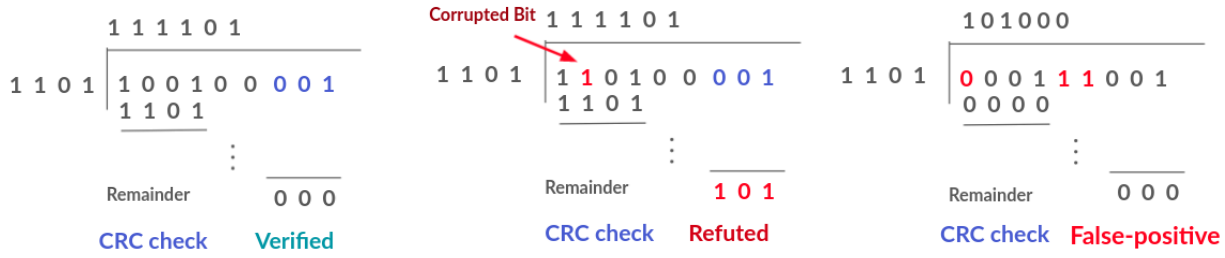


Figure 2.13: The procedure of diving the data (100100) by the CRC divisor to compute the remainder of division.

packets, it uses ACR to estimate the error location. Then it carries out the recovery process segment by segment. To recover a corrupted segment, it starts to permute the values of those segments whose indices are among estimated corrupted locations. It continues the search until the associated ICRC of the segment verifies it.

During these CRC checks, the CRC might incorrectly verify erroneous segments. First, we review how CRC is computed for a block of data, and then we mention an example of false-positive even in CRC checks.

To compute the CRC for a block of data, first, $m-1$ zero bits are appended to the right side of the data, where m refers to the number of bits required to represent the divisor of CRC in binary. Then, data together with appended zero bits is divided by the CRC divisor. This division is based on the arithmetic of a finite field with two elements: 0 and 1. Then, the remainder of this division



(a) Successful verification (b) Successful refutation (c) Inccorect verification

Figure 2.14: (a) The remainder of the error-free version of data is zero; (b) the remainder of erroneous data is not zero; (c) an example of false-positive events in CRC checks when the remainder of erroneous data is zero.

is appended to the original data, and the sender transmits this version of data. Figure 2.13 shows an example of this procedure.

On the receiver side, to check the integrity of data, the received data is divided by the CRC divisor, and the CRC check verifies the integrity of data if the remainder is zero. However, there is a possibility of incorrectly verifying an erroneous block of data. Figure 2.14 gives an example of false-positive events during CRC check. In this example, while the 1st, 5th, and 6th bits of data have changed during transmission, after dividing the received version of data by divisor of the CRC, the remainder is zero, and CRC check incorrectly verifies it. We refer to it as a false-positive event in the CRC check.

Any false-positive event that happens during the correction causes failure in the recovery process. In S-PRAC, as the size of the segment or number of source packets increases, the probability of a false-positive CRC check rises and degrades overall performance and wastes bandwidth.

2.4 Conclusion

PPR schemes that rely on ACR for their estimation phase suffer from common drawbacks. For instance, the estimation phase does not begin for a group of g source packets until the $(g+1)$-th packet is received. This causes the recovery and decoding operations to take longer. Further, when ACR identifies a group of symbols as inconsistent, at least one of those symbols is erroneous. The fraction of inconsistent groups increases as the number of source packets increases, slowing the recovery process and having a more significant impact on packets with more symbols. Moreover, Cabrera et al. demonstrate in [23] that while PRAC is capable of recovering the majority of partial packets, in theory, it has a high computational complexity that is increased for large packet sizes. This also negatively affects the decoding delay. One of the significant benefits of NC communications over other coding techniques is recoding at intermediate nodes [44]. Previously proposed PPR methods only consider point-to-point communications and cannot recover partial packets at intermediate nodes. Also, These partial packets cannot participate in the recoding procedure. As a result, communication cannot take advantage of beneficial aspects of NC.

Chapter 3

Proposed methods

3.1 Introduction

In this chapter, we propose two separate PPR schemes for NC. In the first section, we introduce F-PRNC, which is a point-to-point PPR method for RLNC. F-PRNC aims to reduce the decoding and recovery time, and it is designed to handle burst errors. In the first section, we explain the encoding, recovery, and decoding procedure of F-PRNC. In the second section, we introduce another PPR scheme called Fly-PRAC. Fly-PRAC uses a new estimation technique for identifying corrupted parts of a packet. Also, there are three versions of Fly-PRAC for RLNC, SNC, and relay nodes. In the second section, the Fly-PRAC's estimation method, encoding, recovery, and decoding procedures are explained in detail.

3.2 F-PRNC

The encoding, recovery, and decoding processes of F-PRNC are described in this section.

3.2.1 Encoding

The encoding process is performed on g uncoded packets $\{p_1,p_2,...,p_g\}$, where g is the generation size, and each uncoded packet contains l blocks called symbols $\{s_{i,1}, s_{i,2}, ..., s_{i,l}\}$. All symbols have the same size containing q bits and are chosen from the finite field $GF(2^q)$. The matrix P represents the uncoded packets as a $g \times l$ matrix. In the encoding process, g uncoded packets are transformed into n coded packets using Eq. 3.1, where C is an $n \times g$ coefficient matrix whose elements are selected randomly from $GF(2^q)$.

$$P' = C_{(n\times g)} \times P_{(g\times l)}. \tag{3.1}$$

For each coded packet p'_i containing l coded symbols $\{s'_{i,1},s'_{i,2},...,s'_{i,l}\}$, R' redundancy symbols $\{r_{i,1},r_{i,2},...,r_{i,R'}\}$ are produced using Eq. 3.2 and are appended to p'_i. C' is a coefficient matrix with randomly selected elements from $GF(2^q)$.

$$C'_{(R'\times l)} \times \begin{pmatrix} s'_{i,1} \\ s'_{i,2} \\ \vdots \\ s'_{i,l} \end{pmatrix}_{(l\times 1)} = \begin{pmatrix} r_{i,1} \\ r_{i,2} \\ \vdots \\ r_{i,R'} \end{pmatrix}_{(R'\times 1)} . \tag{3.2}$$

Additionally, one CRC called CRC_s is computed over the coded symbols and attached to p'_i. Each transmitted packet conveys the associated set of coefficients, or both the receiver and the sender synchronize their random coefficient generators. Here, we assume the latter one.

3.2.2 Recovery & Decoding Process

Here we discuss the decoding processes of F-PRNC. F-PRNC decoding process consists of 2 subsequent recovery phases that we call limited and hinted recovery denoted by LR and HR, respectively. The decoding process of F-PRNC is also described in Algorithm 1.

Algorithm 1 F-PRNC decoding process

```
1: Receive the i-th packet (p'_i)
2: result ← limited recovery (p'_i)
3: if result = successful recovery then
4:     add p'_i to valid packets' buffer
5: else
6:     add p'_i to corrupted packets' buffer
7: end if
8: if i ≥ g + 1 then
9:     go to 14.
10: else
11:    i ← i+1 and go to 1.
12: end if
13: while the size of valid packets' buffer < g do
14:    Indices of the inconsistent columns (IC) ← ACR (corrupted packets' buffer , valid packets' buffer)
15:    corrupted packet ← Extract the earliest element of corrupted packets' buffer
16:    result ← hinted recovery (corrupted packet, IC)
17:    if result = successful recovery then
18:        add recovered packet to valid packets' buffer
19:    else
20:        discard corrupted packet, i ← i + 1 and go to 1.
21:    end if
22: end while
23: Run the RLNC decoder with packets in the valid packets' buffer
```

3.2.2.1 Limited recovery

After receiving p'_i, the LR process checks the CRC_s, and if it is correct, it adds p'_i to the valid packets' buffer. Otherwise, it tries to recover the coded symbols of p'_i.

To recover a partial packet, the algorithm tries to find a subset of l error-free symbols out of $l+R'$ coded and redundancy symbols within the packet. Each redundancy symbol is a linear combination of coded symbols within the packets. If the selected subset contains redundancy symbols, using coefficient vectors, l coded symbols can be obtained from it. If the associated CRCs verifies the

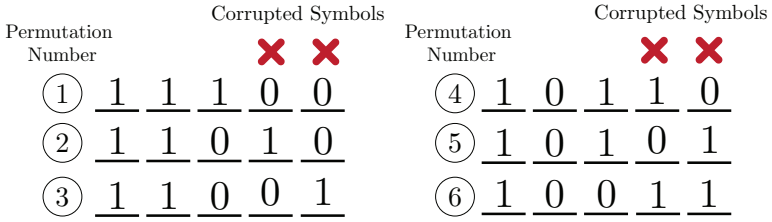


Figure 3.1: The required permutations of V to recover packets containing up to 2 corrupted symbols, where $l = 2$ and $R' = 3$.

subset of selected symbols, the packet is recovered. Otherwise, this process is repeated with another subset. For selecting these subsets, a Boolean vector of size $(l + R')$ is generated, denoted by V in which the R' leftmost elements are ones, and the rest are zeros. The indices of the zero elements in V indicate the subset of l selected symbols. Having corrupted symbols in this subset leads to an incorrect recovery, and CRCs will refute it. If the recovered symbols are incorrect, then V is permuted to the reverse lexicographic order [45], and another subset is selected. The LR process repeats this procedure t times, and if it can find a proper subset, it recovers and adds the packet to the valid packets' buffer, and if it fails, it adds p'_i to the corrupted packets' buffer. t can be considered a threshold for LR and can be an arbitrary constant or set to recover all packets containing corrupted symbols up to T. For packets with T corrupted symbols, by considering the permutation rule applied to V, the packet with T rightmost corrupted symbols is the last packet to be recovered. Consider the example in Figure 3.1. The error locations must have the value "one" to recover the packet, which can be achieved using permutations. According to Figure 3.1, in the first permutation, $\binom{3}{1}$ permutations (permutations 2 to 4) are required to shift the "one" value of the second index to the third index. Generally, at most $\sum_{i=1}^{l} \binom{l+T-i}{T-1}$ permutations are needed to recover all T symbol errors, where $0 \leq T \leq R'$. Whenever the valid packets' buffer size reaches g, the packets are passed

to the RLNC decoder. Otherwise, packets are received until the $(g+1)$-th one.

3.2.2.2 Algebraic consistency rule

ACR in NC has been proposed in [25] and is used to estimate the error locations by checking the consistency of coded packets. Assume that all coded symbols of $g+1$ received packets are stored in $U'_{(g+1)\times l}$. The ACR check performs column by column over matrix U'. For instance, to check the consistency of $j-$th column, the inverse of coefficient matrix is multiplied with $j-$th column of U'. The resulting vector contains estimated symbols $\{s^*_{1,j}, s^*_{2,j}, ..., s^*_{g,j}\}$.

$$\begin{pmatrix} s^*_{1,j} \\ s^*_{2,j} \\ \vdots \\ s^*_{g,j} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,g} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,g} \\ \vdots & \vdots & \ddots & \vdots \\ c_{g,1} & c_{g,2} & \cdots & c_{g,g} \end{pmatrix}^{-1} \times \begin{pmatrix} s'_{1,j} \\ s'_{2,j} \\ \vdots \\ s'_{g,j} \end{pmatrix}. \tag{3.3}$$

The estimated symbols are then re-encoded with the coefficients associated with the $(g+1)$-th packet.

$$s''_{(g+1)j} = \begin{pmatrix} c_{(g+1),1} & \cdots & c_{(g+1),g} \end{pmatrix} \times \begin{pmatrix} s^*_{1,j} \\ s^*_{2,j} \\ \vdots \\ s^*_{g,j} \end{pmatrix}. \tag{3.4}$$

Since the re-encoded and encoded symbols with the same coefficients must be identical in NC, if $s''_{(g+1),j} = s'_{(g+1),j}$ is held, the j-th column is consistent with high probability. Otherwise, the j-th column is inconsistent, i.e., at least one of the coded symbols has been corrupted. Finally, indices

of the inconsistent columns are stored in an array called IC.

3.2.2.3 Hinted recovery

In this process, a packet from the corrupted packet's buffer is selected. The coded symbols that are not present in the inconsistent columns (IC) are then chosen. Suppose that the indices of N_{vs} coded symbols are not included in IC. Similar to the LR process, the HR process generates a Boolean vector called V' to find a subset of $l - Nvs$ non-corrupted symbols. Unlike the LR process, there is no threshold on the number of correction trials in HR. If the HR process fails to find the proper subset, the packet contains more than R' corrupted symbols and cannot be recovered. Therefore, the receiver discards the packet and requests another. Otherwise, it recovers the packet, adds it to the valid packets' buffer, and deletes the corrupted one from the corrupted packets' buffer. Once the number of the valid packets reaches g, the valid packets will be passed to the RLNC decoder.

3.3 Fly-PRAC

Here we discuss the encoding, recovery, and decoding processes of Fly-PRAC.

3.3.1 Fly-PRAC For RLNC

3.3.1.1 Encoding

Assume a set of g original packets $\{p_1, p_2, ..., p_g\}$ are produced at the transmitter node. Each p_i comprised of l blocks called symbols $\{s_{i,1}, s_{i,2}, ..., s_{i,l}\}$ where each symbol is an elements of $GF(2^q)$. In order to encode g original packets into n coded ones, first, we form a matrix using packets and denote it as $P_{(g \times l)}$.

$$P = \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,l} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ s_{g,1} & s_{g,2} & \cdots & s_{g,l} \end{pmatrix}. \tag{3.5}$$

To encode a packet, a coefficient vector with g elements is required. Each element of this vector is selected randomly and independently from $GF(2^q)$. The encoder first generates $R-1$ coefficient vectors. Then form a matrix called the coefficient matrix that contains these coefficient vectors as its rows. Each of these matrices can produce $R-1$ coded packets, and the encoder generates at least $\frac{g}{R-1}$ of such matrices.

Let C^i refer to the i-th coefficient matrix. Each row of C^i comprises a coefficient vector where c^i_j refers to the coefficient vector in the j-th row of C^i.

$$C^i = \begin{pmatrix} c^i_1 \\ c^i_2 \\ ... \\ c^i_{R-1} \end{pmatrix}, \tag{3.6}$$

To form coded packets, the C_i matrix is multiplied with P. The resulting vector is denoted by P'_i and contains $R-1$ coded packets.

$$P'_i = C^i_{(R-1 \times g)} \times P_{(g \times l)}. \tag{3.7}$$

Afterward, a packet that is dependent with $R-1$ previously coded packets is produced. At first, the encoder performs element-wise XOR operation over coefficient vectors of previous $R-1$ coded packets to form a proper coefficient vector.

$$c_R^i = c_1^i \oplus c_2^i \oplus ... \oplus c_{R-1}^i. \tag{3.8}$$

Later, it multiplies c_R^i with P to form a dependent coded packet. We refer to this set of R packets as a group of dependent packets.

$$p'_{iR} = c_R^i \times P. \tag{3.9}$$

Next, each coded packet p'_i containing l coded symbols $\{s'_1, s'_2, ..., s'_l\}$ is divided into s equally-sized segments. Then, for each segment of p'_i, a cyclic redundancy check (CRC) denoted by ICRC is computed over the segment and appended to the coded packets. Another CRC called OCRC is also computed over all coded symbol and is appended to the packet. Therefore, each p'_i contains s segments, s ICRCs and one OCRC. Each coded packets either conveys the associated set of coefficients, or the transmitter synchronizes its random coefficient generators with the receiver, which the latter one is assumed here.

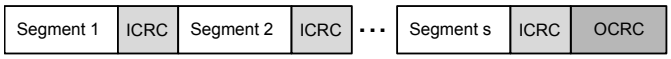


Figure 3.2: Packet format after encoding process.

3.3.1.2 Recovery

Upon reception of a packet, its consistency is checked using the associated $OCRC$. If the $OCRC$ verifies a packet, it's categorized as a valid packet and inserted to B_{valid} buffer. Afterward, it is passed to the decoder. On the other hand, if $OCRC$ refutes a packet, it is categorized as an invalid packet and is inserted into $B_{invalid}$.

Moreover, after receiving R number of packets, if $B_{invalid}$ contains only one packet, this packet will be discarded since, in each R group of packets, one is redundant, thus by recovering it, no new information can be gained. Otherwise, those R packets are selected from $B_{invalid}$ and B_{valid} to recover invalid ones in the recovery process. The objective of the recovery stage is to recover these partial packets using two subsequent processes namely error location estimation and correction by permutation. In this section, we discuss both in detail.

3.3.1.2.1 Error Estimation Step This step intends to provide an estimation of error locations among received packets. Similar to ACR, we use the inter-packet algebraic consistency among packets. Upon reception of all coded packets of the i-th dependent group, the estimation process begins. In the first step, two matrices are formed, namely, C and S. Each row of C contains the coefficient vector of a coded packet in the i-th dependent group.

$$C = \begin{pmatrix} c_{i,1} & c_{i,1} & \cdots & c_{i,g} \\ c_{i+1,1} & c_{i,1} & \cdots & c_{i+1,g} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i+R,1} & c_{i,1} & \cdots & c_{i+R,g} \end{pmatrix}. \tag{3.10}$$

In matrix S, we keep symbols of each coded packet in one row of S. The coefficients associated with the packet in the i-th row of S can be found at the i-th row of C.

$$S = \begin{pmatrix} s'_{i,1} & s'_{i,2} & \cdots & s'_{i,l} \\ s'_{i+1,1} & s'_{i+1,2} & \cdots & s'_{i+1,l} \\ \vdots & \vdots & \ddots & \vdots \\ s'_{i+R,1} & s'_{i+R,2} & \cdots & s'_{i+R,l} \end{pmatrix}. \tag{3.11}$$

While we perform the Gaussian elimination algorithm over C, we do the same row elementary operations on S. The only difference is that the scalar used in the operation of matrix S is provided by matrix C.

In general, if $row(C) < col(C)$ and $rank(C) < row(C)$, then at least one of the coefficient vectors in C is linearly dependent. Therefore, after performing Gaussian elimination over C in one of the rows, all elements must be zero. Assume that after performing Gaussian elimination over C, the i-th row only contains zero elements. If the received coded symbols are error-free, the i-th row of matrix S must contain zero elements. In other words, if we encode a packet with a coefficient vector where all its elements are zero, then all generated coded symbols must be zero too. Since these symbols are sent through a noisy channel, some of them may be corrupted. Thus, the non-zero elements indicate an inconsistency in that column. That means at least one of the elements in that column is corrupted. The process generates a vector called V_{Broken} including the index of detected inconsistent columns. The V_{Broken} then is used in the correction step. The pseudocode of this process is provided in Algorithm 2.

Algorithm 2 Error Location Estimation

$a^c_{i,j}$: the element in the i-th row and j-th column of C
$a^S_{i,j}$: the element in the i-th row and j-th column of S
r^c_i : i-th row of C
r^s_i : i-th row of S
R : number of row in C and S
$P_r = 1$: pivot row initialization
$P_k = 1$: pivot column initialization
index(r_i) : gives the index of packet in the i-th row
while $P_r < R$ and $P_k < g$ **do**
    I $\leftarrow argmax_i(i = P_r \ to \ R, a^c_{i,P_k})$
    if $a^c_{I,P_k} = 0$ **then**
        $P_k = P_k + 1$.
    else
        swap($r^c_{P_r}$,r^c_I)
        swap($r^s_{P_r}$,r^s_I)
        for $i = 1$ to $R - 1$ **do**
            for $j = i + 1$ to R **do**
                scalar $\leftarrow a^c_{j,i}/a^c_{i,i}$
                $r^c_j \leftarrow r^c_j - (scalar \times r^c_i)$
                $r^s_j \leftarrow r^s_j - (scalar \times r^s_i)$
            end for
        end for
    end if
    $P_k = P_k + 1$
    $P_r = P_r + 1$
end while
Index $\leftarrow$ return the index of a row in C where all its elements are zero.
if Index $\neq \emptyset$ **then**
    $V_{Broken} \leftarrow r^s_{Index}$: V_{Broken} contains indices of inconsistent columns
    return V_{Broken}
end if

3.3.1.2.2 Error Correction Step The correction step aims to recover partial packets. For each partial packet, first, it checks the validity of its segments using associated ICRCs. Upon detecting invalid segments of the packet, the operation immediately begins for each invalid segment. Fly-PRAC tries to recover invalid segments by permuting the value of symbols whose indices are contained in V_{Broken}.

After each permutation, the integrity of the segment is checked using associated ICRC. Permutations continue until ICRC verifies the segment. Here, we begin with permutations that have a hamming distance of 1 from the received version of the segment. If there is no verifiable pattern with this hamming distance, the process continues with permutations of bigger hamming distances. In some cases, you might decide to put a threshold on the number of permutations and only try permutations within a specific range of hamming distance from the received packet.

After verifying a segment, the recovery process moves on to the next invalid segment. When all of the invalid segments in a partial packet are recovered, the OCRC is re-computed. If the OCRC verifies the integrity of the packet, the recovered version of the packet is inserted to B_{Valid}; otherwise, it is discarded. Furthermore, since one of the packets in each group of R packets is dependent, the recovery stops whenever $R-1$ packets in the group are recovered or are among valid ones. The pseudocode of this process is provided in Algorithm 3.

3.3.1.3 Decoding

The decoder keeps packets of B_{Valid} in a matrix form. Each row of this matrix contains coefficients and coded symbols of a packet as its elements. Once a new packet is inserted, the decoder runs Gaussian elimination and keep this matrix on its reduced echelon form. Furthermore, whenever the

Algorithm 3 Fly-PRAC recovery process

correction : this function represents the correction step of Fly-PARC.
Receive the i-th packet (p'_i)
status $\leftarrow$ categorize p'_i as valid or partial using associated OCRC
if *status* = valid **then**
    add p'_i to *valid packets' buffer* (B_{Valid})
    Keep packets in B_{Valid} in a matrix form denoted by M
    Keep M on its reduced echelon form
    pass p'_i to the decoder
else
    add p'_i to *invalid packets' buffer* ($B_{Invalid}$)
end if
if i *mod* $R = 0$ and $B_{Invalid}$ size ≥ 1 **then**
    Dependent Group (D)$\leftarrow$ Select latest R received packets
    $N_i \leftarrow$ store number of invalid packets in D.
    if $N_i = 1$ **then**
        Discard the invalid packet in D from $B_{Invalid}$.
    else if $N_i \geq 1$ **then**
        $V_{Broken} \leftarrow$ Estimate Error locations of D
        $D_{invalid} \leftarrow$ Invalid packets in D
        Recovered Packets $\leftarrow$ correction ($D_{invalid}$, V_{Broken})
        Add the recovered packets to B_{Valid}
        Discard packets in $D_{invalid}$ from $B_{Invalid}$
        Pass the recovered packets to the decoder
    end if
end if
if Rank of $M = g$ **then**
    stop the recovery.
end if

rank of matrix reaches g. it decodes all coded symbols and finishes the process.

3.3.2 Fly-PRAC for Relay nodes

The ability to recoding packets at an intermediate node is one of the most beneficial aspects of NC. However, partial packets cannot participate in recoding. This is due to the lack of information needed in recomputing the ICRCs and OCRC. Figure 3.3 represents the percentage of partial packets after passing through a different number of relays. For instance, after crossing two relays connected with communication channels, each with BER of 10^{-4}, roughly 90% of packets are partial.

In an end-to-end packet recovery solution, these partial packets, either are discarded or forwarded. In case of discarding such packets, the maximum throughput degrades with the rate of $\prod_{i=1}^{r}(1-e_i)$ where r indicates the number of relays and e_i refers to error rate of channel between the i-th and

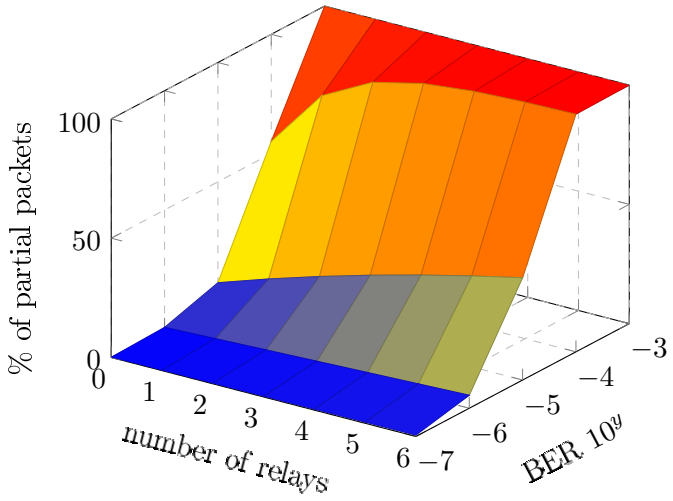


Figure 3.3: Percentage of partial packets after passing through a different number of relays, where the size of the coded symbol is 1000 bytes.

$(i+1)$-th nodes. If relay nodes forward partial packets, the received packets will contain more and more corrupted symbols leading to an increase in the recovery time. To resolve such issues, we propose Algorithm 4, to enable recoding and recovery at intermediate nodes. Figure 3.4 compares forwarding partial packets with the proposed method.

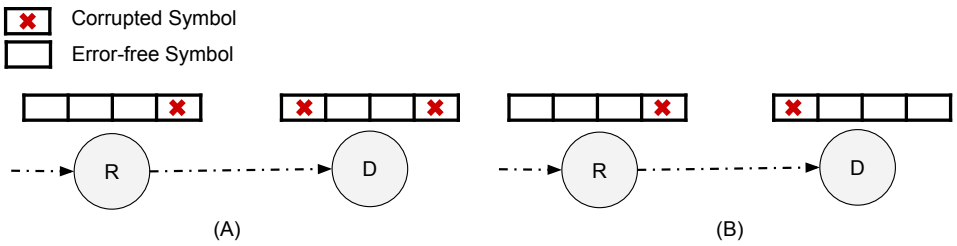


Figure 3.4: In (A) the relay node (R) works in a store-and-forward mode and transmits partial packets; In (B) relay node (R) recovers the packet and then transmits a recoded packet using received and recovered ones.

Algorithm 4 Relay with enabled recoding and recovery

Receive the i-th packet (p'_i)
status $\leftarrow$ categorize p'_i as valid or partial using associated OCRC
if *status* = valid **then**
    add p'_i to *valid packets' buffer*
    send a recoded packet using packets in *valid packets' buffer*
else
    add p'_i to *partial packets' buffer*
end if
if i *mod* $R = 0$ and *partial packets' buffer* is not empty **then**
    recovered packets' buffer $\leftarrow$ recover packets in partial packets' buffer using Fly-PRAC
    $N_r \leftarrow$ Number of packets in recovered packets' buffer
    $N_v \leftarrow$ Number of packets in valid packets' buffer
    Recode N_r packets using packets in valid and recovered packets' buffer and send them.
    if $R - N_v - N_r > 0$ **then**
        generate $R - N_v - N_r$ recoded packets using valid and recovered packets' buffer and send them
    end if
    Discards all packets in recovered, partial and valid packets' buffer
end if

3.3.3 Fly-PRAC for Sparse network coding

SNC is an implementation of NC. One of the main advantageous features of this implementation is the ability to decode some of the packets (i.e., partial decoding) before receiving the entire packets within a generation. This feature is beneficial to a range of applications, such as real-time video transmission.

The main difference between SNC and RLNC is the sparsity of coefficient vectors. While RLNC chooses the elements of the coefficient vectors at random from a finite field, in SNC, most of the elements in the coefficient vectors are zero. Apart from its benefits, this feature leads to the generation of a significant number of dependent packets [35].

Here we propose a variation of the Fly-PRAC algorithm for SNC-based communications that take advantage of those dependent packets in its recovery process. In the following, the encoding, recovery, and decoding processes of this version of Fly-PRAC are discussed.

3.3.3.1 Encoding

The encoding process of Fly-PRAC for SNC and RLNC is quite similar. The only difference between these two in the encoding procedure is the generation of coefficient vectors.

Regarding the SNC scheme, coefficient vectors are w-sparse. This means only w elements of the coefficient vector are non-zero. The positions of w elements are randomly chosen. Also, each of w non-zero elements is randomly and independently selected from $GF(2^q) - \{0\}$. The rest of the encoding process is similar to the encoding process of Fly-PRAC for RLNC.

3.3.3.2 Recovery and Decoding

After receiving a packet, it is categorized as an invalid or valid packet using the associated $OCRC$. If the packet is valid, then it is inserted into a buffer denoted by B_{valid} and passed to the decoder. However, if the received packet is invalid, it is inserted into another buffer denoted by $B_{invalid}$.

Let P'' be a set of packets in both $B_{invalid}$ and B_{valid} where $P'' = \{p''_1, p''_2, ..., p''_m\}$ and $p''_i = \{c_{i,1}, ..., c_{i,g}, s'_{i,1}, ..., s'_{i,l}\}$. We constitute a matrix using packets in M'.

$$M' = \begin{pmatrix} p''_1 \\ p''_2 \\ ... \\ p''_m \end{pmatrix}. \tag{3.12}$$

Fly-PRAC tries to find a group of dependent packets in matrix M'. First, it dedicates one vector to each coded packets in M' where V_i refers to the vector relating to P''_i. Then tries to obtain the reduced echelon form of M' by performing the Gaussian elimination algorithm. During the Gaussian

elimination, every time a packet is involved in the row elementary operation of other packets, its index is inserted into their corresponding vectors. Once the Gaussian elimination process ends, in the case of any group of dependent packets, one of the rows in M' would be a row where its first g elements are zeros. Assume the i-th row of M' has this characteristic. Similar to the estimation process of Fly-PRAC for RLNC, the last l elements of the i-th row must be zero; otherwise, it is evidence of the error for a group of packets whose indices are in V_i. The pseudocode of this process is provided in Algorithm 5.

Algorithm 5 Finding Dependent Group

```
r_i : i-th row of E
a_{i,j} : the element in the i-th row and j-th column
m : number of row in E
P_r = 1 : pivot row initialization
P_k = 1 : pivot column initialization
index(r_i) : gives the index of packet in the i-th row
while P_r < m and P_k < g do
    I ← argmax_i(i = P_r to m, a_{i,P_k})
    if a_{I,P_k} = 0 then
        P_k = P_k + 1.
    else
        swap(r_{P_r}, r_I)
        for i = 1 to m − 1 do
            for j = i + 1 to m do
                scalar ← a_{j,i}/a_{i,i}
                if scalar ≠ 0 then
                    r_j ← r_j − (scalar × r_i)
                    θ ← index(r_i)
                    γ ← index(r_j)
                    V_θ ← V_θ ⋃ V_γ
                end if
            end for
        end for
    end if
    P_k = P_k + 1
    P_r = P_r + 1
end while
ANS ← return a row where its first g elements are zero
if ANS ≠ ∅ then
    I ← Index(ANS)
    D ← V_I ⋃ {I} // D: indices of packets in dependent group
    L ← last l elements of ANS // L: estimation of corrupted locations for packets in D
    return (D, L)
end if
```

If among packets involved in the dependent group, no one is invalid, one of the valid packets

within the group is discarded from B_{valid} since it is redundant. If only one of the packets in the dependent group is invalid, then it is discarded from $B_{invalid}$ since it is also redundant and broken. However, if there are two or more invalid packets, the recovery process begins to recover these packets. Once g number of valid or recovered packets are in B_{valid}, the decoder can decode all source packets in the generation. Algorithm 6 shows the pseudocode of this process.

Algorithm 6 Fly-PRAC for SNC

```
1: correction : this function represents the correction step of Fly-PARC.
2: Receive the ith packet (p'_i)
3: status ← categorize p'_i as valid or partial using associated OCRC
4: if status = valid then
5:     add p'_i to valid packets' buffer (B_Valid)
6:     Keep packets in B_Valid in a matrix form denoted by M
7:     Keep M on its reduced echelon form
8:     pass p'_i to the decoder
9: else
10:    add p'_i to invalid packets' buffer (B_Invalid)
11: end if
12: D : dependent group, L : Broken Locations ← DependentGroup(B_Invalid ⋃ B_Valid)
13: if D ≠ ∅ then
14:    N_i ← store number of invalid packets in D
15:    if N_i = 0 then
16:        Stop the recovery process.
17:    else if N_i = 1 then
18:        Discard the invalid packet in D from B_Invalid.
19:        Stop the recovery process.
20:    else if N_i ≥ 1 then
21:        V_Broken ← Estimate inconsistent columns of D
22:        D_invalid ← Invalid packets in D
23:        Recovered Packets ←correction (D_invalid, V_Broken)
24:        Add the recovered packets to B_Valid
25:        Discard D_invalid from B_Invalid
26:        Pass the recovered packets to the decoder
27:    end if
28: end if
29: if Rank of M = g then
30:    stop the recovery.
31: end if
```

Chapter 4

Analysis

4.1 Introduction

In the first section of this chapter, we analyze the complexity of the F-PRNC scheme in the encoding and decoding section. We also calculate the expected number of correction trials at limited and hinted phases of the recovery process.

In section 2, alongside investigating the complexity of the decoding process of Fly-PRAC, we study the impact of CRC types and the size of the dependent group (R) on false positive events and the performance of the recovery process.

4.2 F-PRNC

Here we discuss the complexity of the recovery and decoding processes of F-PRNC.

4.2.1 Limited Recovery Phase

Let e be the probability of a symbol being corrupted and $L = l + R'$. The probability of having i corrupted symbols in a packet is equal to

$$P_e(i) = e^i(1-e)^{L-i} \binom{L}{i}, \tag{4.1}$$

where e refers to symbol error rate. Suppose φ_i is the set of all $\binom{L}{i}$ possible locations of i symbol errors in a packet, and φ_i^j gives the jth possible error from φ_i. Moreover, $\varphi_i^j(m)$ indicates the mth error-index, stored in the descending order in φ_i^j, where $1 \leq m \leq i$. Suppose that V^y is the yth element of V. Consider a packet with φ_i^j error locations, where $i \leq R'$. The number of permutations to recover such packets is obtained by the sum of required permutations to shift the rightmost "one" to the error locations until $V^{m'} = 1$ for all $m' \in \varphi_i^j$. If the error position is to the left of the rightmost non-shifted "one", no permutation is required. $\sum_{X=x_1}^{x_2-1} \binom{L-X}{\alpha}$ permutations are needed to shift a "one" from V^{x_1} to V^{x_2}, where $0 < x_1 < x_2 \leq L$ and there are α "one" elements from V^{x_1} to V^L. For this type of analysis, we derive Eq. 4.2 which gives the number of permutations for recovering the corrupted packets.

$$I(\varphi_i^j) = \begin{cases} \sum_{m=1}^{i} \lambda(m) \sum_{X=R'-m+1}^{\varphi_i^j(m)-1} \binom{L-X}{m-1} & i \leq R' \\ \\ \binom{L}{R'} & \text{otherwise,} \end{cases} \tag{4.2}$$

where

$$\lambda(m) = \begin{cases} 1 & \varphi_i^j(m) > R' - m + 1 \\ 0 & \text{otherwise.} \end{cases}$$

Due to the threshold (t) in LR, any packet will be recovered in t or less than t permutations, or the LR process will be terminated. The expected number of permutations for a packet in LR is

$$E_{pL} = \sum_{i=0}^{L} P_e(i) \times \frac{\sum_{j=0}^{\binom{L}{i}} min(I(\varphi_i^j), t)}{\binom{L}{i}}. \tag{4.3}$$

4.2.2 Hinted Recovery Phase

Regarding the number of coded and redundancy symbols, all packets with up to R' corrupted symbols can be recovered by F-PRNC. Here, we refer to packets with more than R' corrupted symbols as unrecoverable (UR) packets. If the LR process recovers packets with up to T corrupted symbols, the remaining partial packets recovered in HR have more than T but fewer than $R' - 1$ corrupted symbols. Such packets are referred to as HDR packets.

The probability of having i corrupted symbols in packets processed in HR, given that the coded packet has at least $T + 1$ corrupted symbols is

$$P(i|i > T) = \frac{\binom{L}{i}\ e^i\ (1-e)^{L-i}}{\sum_{j=T+1}^{L} \binom{L}{j} e^j (1-e)^{L-j}}. \tag{4.4}$$

Moreover, packets that can be recovered in the HR phase are denoted as HDR packets. The

probability that an arbitrary symbol of any HDR packet being corrupted is given by

$$e'_{HDR} = \sum_{i=T+1}^{R'} P(i|i > T)\frac{i}{L}. \tag{4.5}$$

Similarly, the probability of an arbitrary symbol of a UR packet being corrupted is

$$e'_{UR} = \sum_{i=R'+1}^{L} P(i|i > T)\frac{i}{L}. \tag{4.6}$$

Let N_{HDR} and N_{UR} be the number of HDR and UR packets, respectively. The probability that the random symbol of either the UR or HDR packets will be corrupted is denoted by e' and is equal to

$$e' = \frac{N_{HDR}}{N_{HDR} + N_{UR}} e'_{HDR} + \frac{N_{UR}}{N_{HDR} + N_{UR}} e'_{UR}. \tag{4.7}$$

The probability of a random column being erroneous is

$$P_{IC} = 1 - (1 - e')^{N_{HDR}+N_{UR}}. \tag{4.8}$$

Therefore, the expected number of the inconsistent columns will be

$$E_{IC} = \sum_{i=0}^{l} i \binom{l}{i} (P_{IC})^i (1 - P_{IC})^{l-i}. \tag{4.9}$$

Let ϑ_i give all possible locations of the corrupted symbols with i errors in V', and ϑ_i^j give the j-th variation out of $\binom{N_{IC}+R'}{N_{IC}}$. $\vartheta_i^j(m)$ represents the mth element of ϑ_i^j sorted in the descending order.

Let $L' = N_{IC} + R'$. Similar to Eq. 4.2, the number of permutations in HR for recovering a packet with ϑ_i^j error locations in V' is

$$I'(\vartheta_i^j) = \begin{cases} \sum_{m=1}^{i} \lambda'(m) \sum_{X=R'-m+1}^{\vartheta_i^j(m)-1} \binom{L'-X}{m-1} & i \leq R' \\ \\ \binom{L'}{N_{IC}} & \text{otherwise,} \end{cases} \tag{4.10}$$

where

$$\lambda'(m) = \begin{cases} 1 & \vartheta_i^j(m) > R' - m + 1 \\ \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the expected number of permutations for a packet in HR is equal to

$$E_{pH} = \sum_{i=T+1}^{L} P(i|i > T) \times \frac{\sum_{j=1}^{\binom{L'}{i}} I'(\vartheta_i^j)}{\binom{L'}{i}}. \tag{4.11}$$

4.2.3 Complexity of RLNC and ACR

The RLNC decoder uses Gaussian elimination for the decoding process. The complexity of Gaussian elimination over $g+1$ packets is equal to $O(g)^3$. In the ACR process, each column must be multiplied by the inverse matrix of coefficients to obtain the estimated data symbols. The cost of this procedure is also $O(g^3)$.

4.3 Fly-PRAC

Generally, the proposed method on the recipient side consists of recovery and decoding processes. The recovery process passes only valid packets to the decoder, and it ends whenever there are g

independent packets. Also, the decoder performs a Gaussian elimination over packets in B_{valid} to decode symbols. However, the decoder might receive some dependent packets too. For decoding packets, the standard RLNC decoder is employed which has a complexity of $O((g + N_{dp})^3)$ where N_{dp} indicate to the number of dependent packets in B_{valid}.

Besides, the recovery process works on both partial and valid packets, and its complexity depends on the number of received packets. Some partial packets might not be recovered during the recovery process due to the false-positive events in error estimation or correction. Parameters such as generation size (g), field size ($GF(2^q)$), segment numbers (s), and R affects the probability of such events. Therefore, they have an impact on the performance of the proposed method. Here we investigate the impact of these parameters in detail.

4.3.1 False Positive in correction phase

Generally, CRC is a compelling solution to the error detection problem, although it might falsely verify the integrity of erroneous data, which we refer it as a false positive event. According to a given bound for the probability of undetected errors of a CRC in [46], the probability of such event for a segment containing n_s bits and BER of ϵ is denoted by P_{fp} and can be calculated using Eq. 4.12.

$$P_{fp} = \sum_{i=0}^{n_s} A_w^i (1-\epsilon)^{n_s - i} (\epsilon)^i, \tag{4.12}$$

where A_w^i refers to the number of valid codewords with the hamming weight of w. The Eq. 4.12 clearly shows the impact of CRC type and data length on P_{fp}. Assume a packet containing n_p bits, which comprises a segment of size n_s bits and an ICRC with n_c bits where $n_p = n_s + n_c$. We want

to transmit such packets over a noisy binary symmetric channel. Here we also assume the ICRC is error-free. Let V_s and V_s' represent vectors of size n_s, respectively, holding the error-free and received versions of segment data in binary. Also, let E be a vector of size n_s representing errors locations with "1" elements, while other error-free locations are zeros.

$$V_s \oplus E = V_s'. \tag{4.13}$$

Upon receiving a packet, the integrity of information is checked by ICRC, and if it is inconsistent with the data, the recovery process begins. In correction trials, the value of V_s' is changed until the ICRC verifies its integrity. To change the value of V_s', XOR operation is performed on V_s' and V_p. V_p is a vector of the same size as V_s' containing zero and one elements. After each change, associated ICRC checks the integrity of resulting vector V_s''.

$$V_s' \oplus V_p = V_s''. \tag{4.14}$$

Afterward, if ICRC verifies the resulting vector (V_s'') as an error-free version of V_s', the correction ends, and V_s'' is replaced with V_s' in the packet. On the other hand, if the ICRC refutes integrity of V_s'', this process repeats with different V_p vectors until the ICRC verifies the V_s''.

Assume we are using a CRC with linearity properties, i.e., the XOR of two valid codewords yields another valid codeword. If E is not a zero vector and $V_p \oplus E$ constitute a valid codeword, the ICRC will verify it. Since $V_s'' \neq V_s$, a false positive event has happened.

Let α_1 and α_2 be hamming weight of E and V_p in turn. Clearly, the weight of $E \oplus V_p$ is ranging from $|\alpha_1 - \alpha_2|$ to $min(n_s, \alpha_1 + \alpha_2)$. Suppose h_d be the hamming weight of $E \oplus V_p$, therefore the

probability of $E \oplus V_p$ become a valid codeword is denoted by P_{valid} and it can be calculated by Eq. 4.15.

$$P_{valid} = \frac{A_w^{h_d}}{\binom{n_p}{h_d}}. \tag{4.15}$$

Assume a V_p with the weight of α_2 XOR with V_s' containing α_1 bit errors. Suppose j is the number of elements in V_p that are "1" at elements where their corresponding elements in V_s' are erroneous. As a result, the hamming distance between V_s'' and V_s would be $\alpha_1 + \alpha_2 - 2j$. Clearly, the value of j and $\alpha_1 + \alpha_2 - 2j$ can not be more than the size of the segment which is n_s. Also, we refer the minimum value of j as j_{min} which is calculated by Eq. 4.16.

$$j_{min} = \begin{cases} 0 & \alpha_1 + \alpha_2 - n_s \leq 0 \\ \alpha_1 + \alpha_2 - n_s & O.W. \end{cases} \tag{4.16}$$

The recovery algorithm starts by using all permutations of V_p with minimum weight, and obviously, the correct answer is found where $\alpha_1 = \alpha_2$. Thus, α_2 is increased from 1 to α_1. Also, there are $\binom{\alpha_2}{j}\binom{n_s-\alpha_2}{\alpha_2-j}$ possible correction trials for a V_p of weight α_2.

Let P_{nq} indicate the probability of no false positive event during all correction trials using V_p of weight α_2 where $\alpha_1 \neq \alpha_2$. It can be calculated using the Eq. 4.17.

$$P_{nq} = \prod_{j=j_{min}}^{\alpha_2} (1 - \frac{A_w^{\alpha_1+\alpha_2-2j}}{\binom{n_p}{\alpha_1+\alpha_2-2j}})^{\binom{\alpha_2}{j}\binom{n_s-\alpha_2}{\alpha_2-j}}. \tag{4.17}$$

Furthermore, let P_{eq} indicate the probability of no false positive event when during all correction

trials using V_p of weight α_2 where $\alpha_1 = \alpha_2$. The value of P_{eq} can be calculated usint the Eq. 4.18.

$$P_{eq} = \prod_{j=j_{min}}^{\alpha_2 - 1} (1 - \frac{A_w^{\alpha_1+\alpha_2-2j}}{\binom{n_p}{\alpha_1+\alpha_2-2j}})^{\binom{\alpha_2}{j}\binom{n_s-\alpha_2}{\alpha_2-j}}. \tag{4.18}$$

In all, the probability of no false positive event, where E has a hamming weight of α_1, can be calculated using Eq. 4.19 and is denoted by P_{nfp}.

$$P_{nfp} = \prod_{\alpha_2=0}^{\alpha_1} f(\alpha_1, \alpha_2), \tag{4.19}$$

where

$$f(\alpha_1, \alpha_2) \begin{cases} P_{nq} & \alpha_1 < \alpha_2 \\ P_{eq} & \alpha_1 = \alpha_2 \neq 0 \\ 1 & \alpha_1 = \alpha_2 = 0 \end{cases}.$$

Lastly, for a channel with BER of ϵ, the probability of no false positive event or successful recovery of a segment is denoted by P_{sr} and is calculated by Eq. 4.20.

$$P_{sr} = \sum_{\alpha_1=0}^{n_s} (1-\epsilon)^{n_s-\alpha_1}(\epsilon)^{\alpha_1}(\prod_{\alpha_2=0}^{\alpha_1} f(\alpha_1, \alpha_2)). \tag{4.20}$$

Figure 4.1 shows the probability of successful recovery for different BERs. The red and blue trends are P_{sr} calculated using the Eq. 4.20 and the green and black trends are simulation results. This figure illustrates that as BER increases, the P_{sr} falls, and this fall is more dramatic for larger segment sizes.

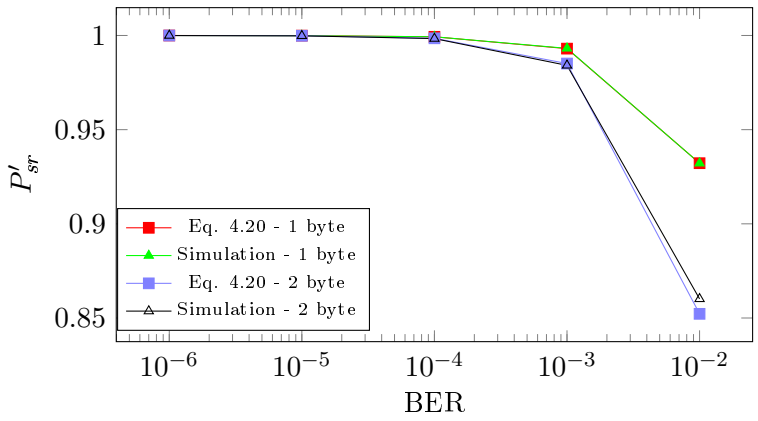


Figure 4.1: The probability of successful recovery of a segment ($P'sr$) for CRC 0xe7 and different BERs.

4.3.2 Impact of R on the recovery process

In this subsection, we investigate the impact of the number of packets in a dependent group (R) on the expected inconsistent columns and false positive events in the estimation process.

For a channel with BER of ϵ, the probability that a received symbol of size q bits is error-free can be calculated by the Eq. 4.21, and it is denoted by p_h^s.

$$p_h^s = (1-\epsilon)^q. \tag{4.21}$$

In the estimation of error locations for a group of R packets, one consistency check is performed over each column. The probability of an arbitrary selected column be error-free is computed by Eq. 4.22 and denoted by $p_h^c(R)$.

$$p_h^c(R) = (p_h^s)^R. \tag{4.22}$$

According to [25], $g+1$ packets are required in ACR process. Conversely, the proposed estimation of error locations rely on a group of R dependent packets where $2 < R \leq g+1$. Therefore $(p_h^c)^R \geq (p_h^c)^{g+1}$. In other words, as the value of R decreases, the probability of receiving an error-free column rises.

Further, the expected number of inconsistent column (E_{ic}) for a group of R packets each with l symbols and s segments is calculated by Eq. 4.23. Figure 4.2 shows a comparison of E_{ic} between proposed estimation process with different R (red and blue trends) and ACR (black trend).

$$E_{ic} = \sum_{i=1}^{l} \binom{l}{i} \times i \times (p_h^c)^{l-i}(1-p_h^c)^i. \tag{4.23}$$

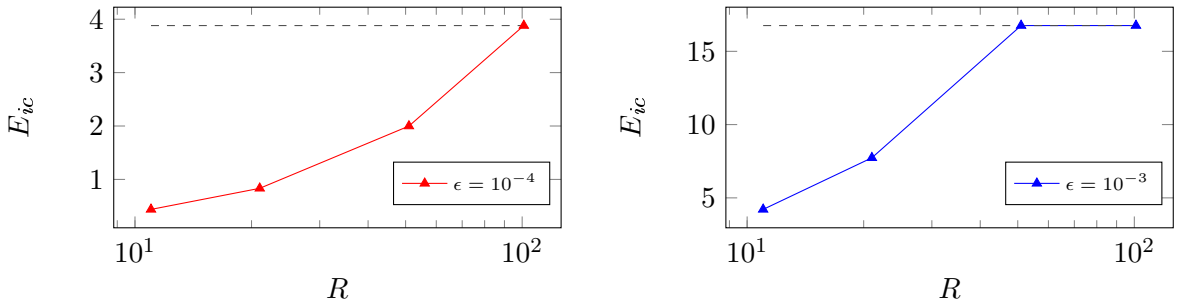


Figure 4.2: The comparison of expected number of inconsistent column for different R (colored) with ACR (Black) in a channel with BER of ϵ where $g = 100$, $l = 50$, and field size is $GF(2^8)$

A fall in $p_h^c(R)$ negatively affects the recovery process by raising E_{ic} and consequently raising the search space. The recovery process performs over each invalid segment by permuting the value of symbols in inconsistent columns. For a symbol with the size of q bits, there are 2^q permutations. Assume the expected number of inconsistent columns for each segment is equal to $\frac{E_{ic}}{s}$. The maximum number of permutations for an invalid segment with $\frac{E_{ic}}{s}$ symbols in inconsistent columns is denoted

by N_p and calculated by Eq. 4.24.

$$N_p = (2^q)^{\frac{E_{ic}}{s}}. \tag{4.24}$$

In addition, figure 4.3 compares the number of involved symbols in inconsistent columns for $R = 5$ and $R = 9$. It illustrates that for the same error pattern, the number of symbols estimated as corrupted ones for $R = 5$ is half of what is for $R = 9$. A reduction in the number of symbols estimated as corrupted ones result in lower search space and can boost the recovery performance.

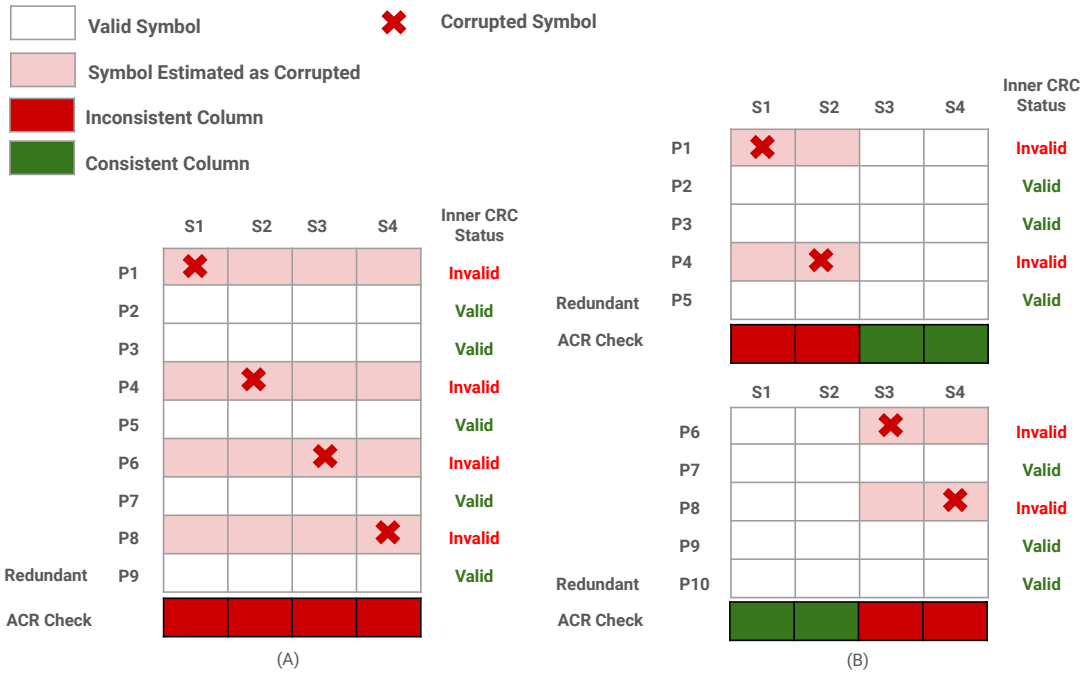


Figure 4.3: For a segment containing 4 symbols and $g = 8$; for Fly-PRAC with (A) $R = 9$ and (B) $R = 5$.

According to [25], when a column is inconsistent, the probability of a false positive event during ACR is equal to 2^{-q} where q refers to the length of each symbol. Here we investigate the probability

of false positive events during the error locations' estimation process.

The false positive event for a column only happen when there are more than one erroneous symbol. In order to illustrate this, first, let D refer to a group of R error-free packets that are linearly dependent.

$$D = \{p'_1, p'_2, ..., p'_R\}. \tag{4.25}$$

Since packets in D are linearly dependent, there exists a set of coefficients (X) with R elements from $GF(2^q) - \{0\}$ that satisfies the Eq. 4.27.

$$\exists X, X = (x_1, x_2, ..., x_R), \forall x_i \in X, x_i \in GF(2^q) - \{0\},$$

where

$$\sum_{i=0}^{R} x_i p'_i = \{0, 0, ..., 0\}. \tag{4.26}$$

Since operations over packets happen symbol-wisely, thus Eq. 4.27 can be rewritten only for the j-th symbol across R packets.

$$\exists X, X = (x_1, x_2, ..., x_R), \forall x_i \in X, x_i \in GF(2^q) - \{0\},$$

where

$$\sum_{i=0}^{R} x_i s'_{i,j} = 0. \tag{4.27}$$

Assume only the j-th symbol of the i-th packet in D is erroneous.

$$s'_{i,j} = s_{i,j} + e, \tag{4.28}$$

where $e \in GF(2^q) - \{0\}$. Also, the rest of symbols $s'_{i,k} = s_{i,k}$ where $k \in \{1, 2, ..., R\} - \{j\}$. Since the j-th symbol is erroneous, the Eq. 4.27 can be rewritten as Eq. 4.29.

$$\sum_{i=0}^{R} x_i s_{i,j} + x_i e = 0 + x_i e. \tag{4.29}$$

The inconsistency of a column will not be detected if the result of Eq. 4.29 is zero. Since both x_i and e are non-zero, the Eq. 4.29 is not zero too. Therefore, no false positive can happen if only one of the symbols in a column is broken.

Similar to the false positive probability of ACR checks, when there exists more than one erroneous symbol in a column, the probability of a false positive event for a column is 2^{-q} [25]. Also, the Eq. 4.30 computes the probability of a false positive event during the estimation of error locations according to the channel condition, which is denoted by p'_{fp}.

$$p'_{fp} = \sum_{i=2}^{R} \binom{R}{i} \times (p_h^s)^{R-i} \times (1 - p_h^s)^i \times \frac{1}{2^q}. \tag{4.30}$$

To provide estimation for an entire generation of g packets, this process must be done for at

least $\frac{g}{R-1}$. There are l columns for each group of dependent packets. For an entire generation of source packets, the consistency of at least $\frac{g}{R-1} \times l$ column must be checked. The expected number of false-positive events for an entire generation is denoted by E_{fp} and is calculated by Eq. 4.31. Further, figure 4.4 shows that a rise in the value of R increases the expected number of false-positive events.

$$E_{fp} = \frac{g}{R-1} \times l \times p'_{fp}. \tag{4.31}$$

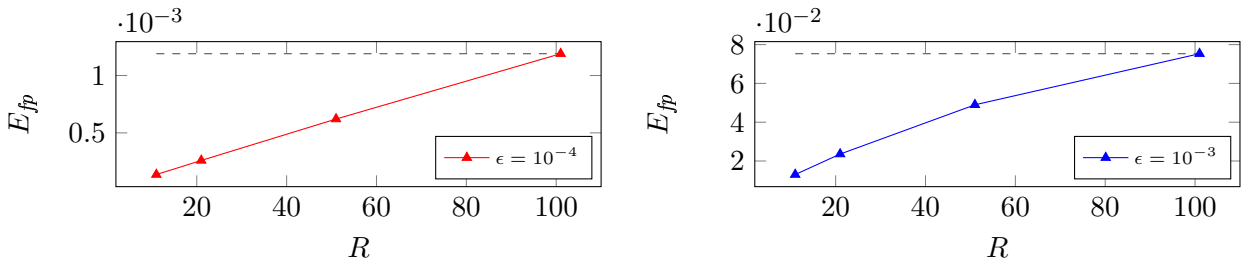


Figure 4.4: The comparison of expected number of false positive events between the ACR (Black) and the proposed error locations' estimation process (colored) for a channel with BER of ϵ where $g = 100$, $l = 50$, and field size is $GF(2^8)$.

For each dependent group, one Gaussian elimination algorithm is performed, which has a complexity of $O(R^3)$. Since there are $\frac{g}{R-1}$ dependent groups for a generation, this complexity for entire generation is equal to $O(\frac{g}{R-1}R^3)$.

Chapter 5

Simulation Results

5.1 Introduction

This chapter compares the proposed methods with the most relevant PPR schemes by carrying out extensive simulations. In the first section, we compare F-PRNC with DAPRAC and S-PRAC in different criteria. We expose packets to burst errors and evaluate the performance of PPR schemes for different byte error rates (ByER). In section two, we compare Fly-PRAC with S-PRAC in a variety of settings. We expose packets to different bit error rates(BER) and evaluate the effectiveness of Fly-PRAC and S-PRAC in different environments. It is essential to mention that the implementation of some methods and definition of some terms between the two sections are slightly different, which we discuss in experiment setups of each section.

5.2 F-PRNC

5.2.1 Experiment Setup

The discussed methods are implemented using the Kodo library [47]. Simulations are performed using $GF(2^8)$, carried out 5000 times, and the average is reported. Moreover, CRC and redundancies are not considered in the payload size. All the mentioned methods use one CRC-32[1] for each coded packet. However, S-PRAC adds an extra CRC-8[2] for each segment. We introduce a goodput as the successful decoding rate of data at the destination. For computing the goodput of F-PRNC, a retransmission delay of $36ms$ is considered as an estimation for the retransmission delay, which is close to the average round-trip time from Gdansk to Bern and Paris [48]. In implementing S-PRAC, we assume that if any false-positive occurs in CRC checks of correction trials, the recovery procedure for that generation of packets fails, and the sender node must resend all coded packets.

5.2.2 Comparison of F-PRNC with DAPRAC and S-PRAC

Fig. 5.1 compares the goodput of DAPRAC, S-PRAC, and F-PRNC, where the byte error rate (ByER) is 0.1, and the payload size is 8 bytes. Fig. 5.1 shows that by increasing the generation size, the goodput of DAPRAC and S-PRAC drops dramatically and almost reaches zero. In DAPRAC, increasing the generation size results in more decoding failures since any error in the exact location of separate packets or an error in any inner CRCs can cause a decoding failure, which is more likely to occur for larger generation sizes. In S-PRAC, CRC-8 checks are used for the detection of errors in the recovery process. Since each segment has one CRC-8, it may incorrectly verify the segments

[1] A CRC is called a CRC-32 when its check value is 32 bits long.
[2] A CRC is called a CRC-8 when its check value is 8 bits long.

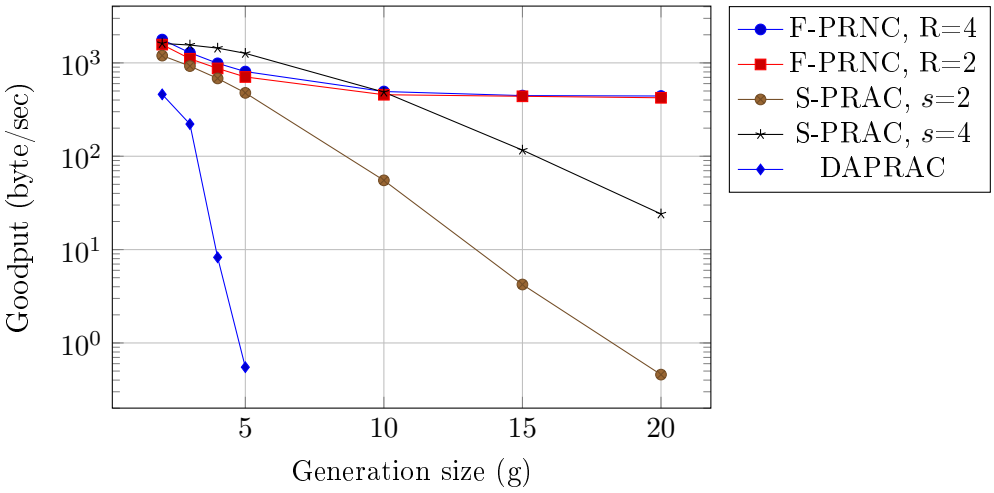


Figure 5.1: The comparison of F-PRNC (with R' redundancy symbols), S-PRAC (with s segments) and DAPRAC, on a log-normal scale, where the payload size is 8 bytes and ByER is 0.1.

with more than 8 bits. This leads to a decoding failure, which is more likely to occur for larger generation sizes. By increasing the generation size, the goodput of F-PRNC decreases, although it has a minor effect after a certain point.

Table 5.1 shows the impact of using redundancy on the decoding and retransmission delays of the extra coded packets, where $36ms$ delay is considered for each retransmission, and ByER is 0.14. If S-PRAC and DAPRAC fail in decoding a generation properly, the generation must be retransmitted until the successful recovery. We also assume that no errors in S-PRAC could occur in the inner CRCs. As shown in Table 5.1, F-PRNC outperforms DAPRAC, and S-PRAC in terms of retransmission and decoding delays in most settings. By increasing the payload size from 8 to 16 bytes and the generation size from 4 to 16, the retransmission and decoding delays increase dramatically for S-PRAC and DAPRAC which make them impractical for real-world applications.

Table 5.1: The comparison of F-PRNC (with R' redundancy symbols), S-PRAC (with s segments), and DAPRAC in terms of retransmission delay (RD) and decoding delay (DD) in milliseconds, where the ByER is 0.14.

Payload size (bytes)			8		16	
Generation size			4	16	4	16
F-PRNC	R'=2	RD	6.6	72.36	34.9	511.2
		DD	71.2	486.5	256.4	4286.2
	R'=4	RD	0.8	8.28	11.23	95.4
		DD	60.8	490.9	406.5	11618.2
S-PRAC	s=2	RD	1507.3	$> 10^6$	143856	$> 10^6$
		DD	143.2	$> 10^6$	13349.7	$> 10^6$
	s=4	RD	367.8	$> 10^6$	21032.5	$> 10^6$
		DD	43.6	34372.6	1944.4	$> 10^6$
DAPRAC		RD	4219.6	$> 10^6$	$> 10^6$	$> 10^6$
		DD	80240.3	$> 10^6$	$> 10^6$	$> 10^6$

5.3 Fly-PRAC

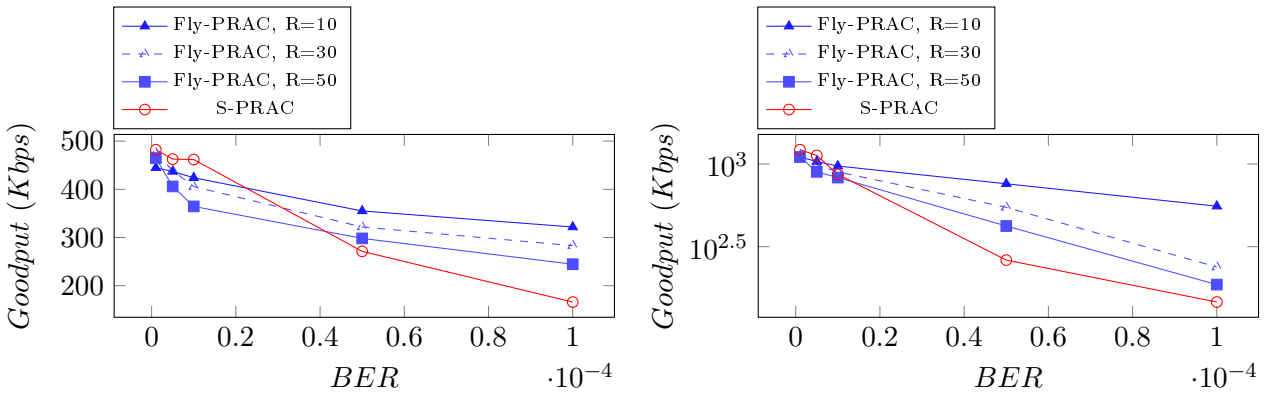
In this section, we compared the performance of Fly-PRAC with its most notable related works, i.e., S-PRAC for various settings. First, in a point-to-point scenario using RLNC for encoding and decoding packets, the effect of generation size and payload size on goodput is reported for a variety of error rates. Further, in a unicast communication through a relay, the impact of enabling recoding and recovery for the intermediate node is investigated. Next, for a point-to-point scenario, the efficiency of packet recovery in terms of goodput and average decoding delay (ADD) for SNC-based communication is also examined. Finally, these methods are compared in terms of recoverability and overhead.

5.3.1 Experiment Setups

All methods are implemented using the KODO library, and each simulation is repeated 10000 times, and the average is reported. Throughout the experiments, we consider a field size of $GF(2^8)$. Further, we assume that only the coded symbols can have errors for a packet, and the rest is immune to errors. In the S-PRAC implementation, we assume that if any false positive events occur during the CRC check correction process, the packet is discarded, and the receiver waits for a sufficient number of coded packets to continue the recovery process.

5.3.2 Impact of Generation and Payload size on Packet Recovery

Here, we have compared the packet recovery performance of Fly-PRAC and S-PRAC in terms of goodput for different BERs. Also, the impact of generation and payload size on completion time is investigated.



(A) Symbol size of 300 bytes

(B) Symbol size of 900 bytes

Figure 5.2: Impact of error rates and payload size on goodput for SPRAC and Fly-PRAC with 4 segments and $g = 100$.

Figure 5.2 shown that for higher BER, the Fly-PRAC outperforms S-PRAS in terms of goodput.

For instance, when payload size is 300 bytes, Fly-PRAC with $R = 10$ has a goodput of 355.1 Kbps compared to 271.2 Kbps for S-PRAC. Further, for larger payload sizes, the Fly-PRAC begins to perform better than S-PRAC at lower error rates. It can be observed that with BER = 10^{-5}, Fly-PRAC with $R = 10$ has a goodput of 972 Kbps compared to 886 Kbps for S-PRAC.

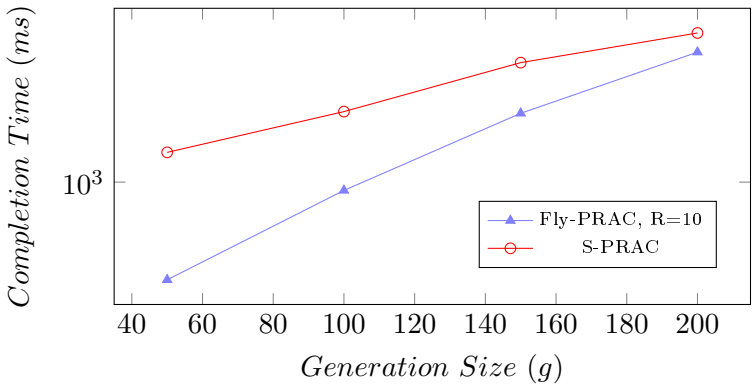


Figure 5.3: Completion time comparison between Fly-PRAC and S-PRAC where BER= $5e-5$ and segment size = 4 and payload size is 800 bytes.

Figure 5.3 illustrates that for larger generation sizes, it takes more time to decode a generation due to the higher complexity of decoding. It can be observed that, Fly-PRAC with $R = 10$, the completion times are 303 ms and 903.7 ms, when the generation sizes are 50 and 100, respectively.

Generally, larger payload sizes have more inconsistent columns relatively, which makes the recovery process more time-consuming. Albeit, Figure 5.4 shows this increase is more dramatic for S-PRAC. For instance, increasing payload size from 300 to 900 bytes leads to an increase in the completion time of S-PRAC from 836.5 ms to 2544.8 ms. While, for Fly-PRAC with $R = 10$ the completion times for the same payload sizes are increased from 667.1 ms to 910.3 ms.

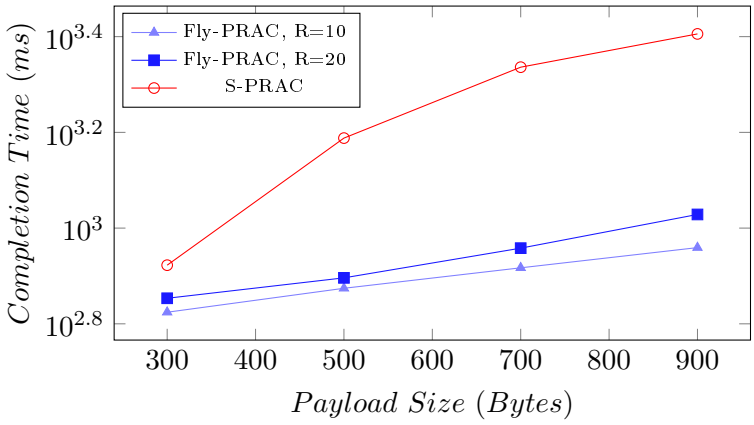


Figure 5.4: Completion time comparison between Fly-PRAC and S-PRAC where $s = 4$, BER= $5e{-}5$ and $g = 100$.

5.3.3 Impact of enabling recovery at intermediate nodes

Here we compare Fly-PRAC and S-PRAC in a unicast scenario where the source and destination communicate through a relay node. The channel between source and relay or relay and destination have a BER of ϵ. For the Fly-PRAC scenario, the relay uses the algorithm 4 i.e., the relay is capable of recoding and recovering partial packets, while the relay in the S-PRAC scenario only works in store-and-forward mode. Table 5.2 clearly illustrates that in a poor channel condition, the proposed method decodes a generation with fewer transmissions. In particular, it is shown that for BER of 5×10^{-5}, Fly-PRAC with R=10 requires 136.6 transmissions on average to decode a generation compared to 146.1 required packets for S-PRAC.

Here in Table 5.2, the time spent by relay, recovering, and recoding packets are also considered in the completion time of Fly-PRAC. It can be observed that for poor channel conditions like BER=10^{-4}, the completion time of Fly-PRAC with R=5 and S-PRAC are 1502 ms and 3069 ms, respectively.

Table 5.2: The comparison of Fly-PRAC and S-PRAC, in terms of Completion Time (CT) and the average sent packets to decode a generation. Both methods using four segments and communicating through a relay where channels have BER of ϵ.

Method	R	ϵ = 5e-5		ϵ = e-4	
		Total Sent	CT(ms)	Total Sent	CT(ms)
Fly-PRAC	5	141.3	1126.2	181.9	1502.5
	10	136.6	1221.9	202	2344
S-PRAC	-	146.1	1476.45	212.3	3069.97

5.3.4 Comparison of packet recovery for SNC communication

Here we evaluated the impact of Fly-PRAC and S-PRAC on partial decoding for SNC. In [35], Zarei et al. suggested a metric called average decoding delay (ADD) to evaluate partial decoding of SNC. ADD is calculated by the following equation.

$$ADD = \frac{\sum_{i=0}^{g} d_i}{g}, \tag{5.1}$$

where d_i indicates the number of transmissions until the ith-packet is decoded.

As we decrease the value of w, more packets must be collected to obtain g innovative packets. For instance, when no partial packet recovery is used, for a channel with BER of 5×10^{-5}, when w is 2 and 3, the average number of required coded packets to decode a generation entirely are 372.7 and 231 in turn.

The proposed method exploits redundant packets to begin recovery earlier. This enhances the ADD of an SNC scheme in noisy conditions. Table 5.3 illustrates that utilizing packet recovery boosts the partial decoding in terms of ADD. For instance, when BER= 5×10^{-5}, $w = 2$ and

Table 5.3: The comparison of Fly-PRAC (with R=5), S-PRAC and No PPR in terms of ADD and average required packet to decode a generation of 100 packets and payload size of 800 bytes.

BER	w		Fly-PRAC	S-PRAC	No PPR
5e-5	2	ADD	103.9	114	113.5
		Total Sent	322.97	326.7	372.7
	3	ADD	118.9	129.8	131.4
		Total Sent	215.9	220.3	231
e-4	2	ADD	106.9	156.7	161
		Total Sent	326.2	427	485.5
	3	ADD	122.6	178	180.5
		Total Sent	220.2	286.6	331.5

$g = 100$, the ADD in case of discarding partial packets is on average 113.5 while using Fly-PRAC reduces it to 103.9. This also leads to a reduction in the number of required transmissions from 372.7 to 322.97 packets.

Moreover, results show that Fly-PRAC is more successful in reducing the ADD compared to S-PRAC. For example, table 5.3 shows when BER= 10^{-5}, $w = 2$ and $g = 100$ the ADD of Fly-PRAC and S-PRAC are 106.9 and 156.7, respectively.

5.3.5 Recoverability and Overhead Comparison

Table 5.4 gives information about average recovered and transmitted packets until successful decoding of a generation for Fly-PRAC and S-PRAC method. In comparison to S-PRAC, Fly-PRAC recovers more packets and requires less transmission in a variety of scenarios. For instance, when $g = 100$ and the payload size is 300 bytes, the Fly-PRAC with $R = 10$ recovers 10 packets compared to 0.9 recovered packets in S-PRAC. Furthermore, Fly-PRAC requires an average of 105.9 packets for successful recovery, whereas S-PRAC requires 112.3 packets. As a result, sending redundant packets

Table 5.4: The comparison of Fly-PRAC and S-PRAC, with segment size 4 and BER = 5e-5.

Method	Generation	R	Payload Size (bytes)			
			300		700	
			Total	Recovered	Total	Recovered
Fly-PRAC	100	5	124.3	5.5	125	20.4
		10	111.2	8.5	112.8	22.9
		15	105.9	10	111.6	19.8
	200	5	249.4	10.8	250.5	41
		10	222.4	16.5	225.5	46.3
		15	211.7	19.7	223	40
S-PRAC	100	-	112.3	0.9	131.7	1.6
	200	-	224.4	1.3	264	1.5

at the proper time reduces the total number of packets. In particular, the overall transmission gap between Fly-PRAC and S-PRAC grows from 12.09 packets for a 300-byte payload to 40.98 packets for a 700-byte payload. However, transmitting too many redundant packets causes more overhead. For instance, for Fly-PRAC with a payload size of 300 bytes, the total number of required transmissions for $R = 5$ and $R = 10$ are 124.3 and 105.9, respectively.

Chapter 6

Conclusion and Future work

Due to the faulty nature of wireless communications, errors are inevitable during transmission. As a result, a considerable number of packets might partially corrupt. In general, partial packets are discarded in network coding communications, which is not optimal and degrades communication performance. In this thesis, we propose two partial packet recovery schemes for network coding to increase reliability and enhance the quality of communication.

First, we proposed a PPR scheme for RLNC called F-PRNC. This scheme intends to reduce decoding and recovery time. F-PRNC benefits from the algebraic consistency check to detect the corrupted symbols that accelerate the decoder's performance. Further, F-PRNC adds redundancy symbols at the end of each coded packet. Once a partial packet is received, F-PRNC exploits appended redundancy symbols to recover partial packets. In this study, we provided the analytical results on the complexity of F-PRNC. We also carried out extensive simulations on its performance. The simulations show that F-PRNC can improve goodput and reduce retransmission delays in very

noisy communication channels. The results demonstrate the superiority of F-PRNC over previously proposed PPR schemes such as DAPRAC and S-PRAC under a variety of conditions.

Moreover, we proposed another PPR scheme called Fly-PRAC. Fly-PRAC uses algebraic consistency between a subset of R packets to estimate the corrupted locations of partial packets. For this purpose, after transmitting $R-1$ packets, the encoder transmits a redundant one. Fly-PRAC can now provide estimation and begin to recover partial packets earlier according to this improved estimation technique. It also reduces the false-positive events during the estimation process. We propose multiple versions of Fly-PRAC for RLNC, SNC, and relay nodes. Fly-PRAC is the first PPR scheme for network coding that allows intermediate nodes to recover partial packets. Therefore, the recovered packets can participate in the re-coding process, allowing communication to utilize NC benefits fully. The simulation results illustrate that, in a poor channel condition, the proposed method can boost the speed of the recovery process and also can recover more packets. These features make it more suitable for larger generation sizes and bigger payload sizes. In sparse network coding communication, we also evaluated the impact of the proposed method on partial decoding. Simulations imply that Fly-PRAC can boost partial decoding by reducing the average decoding delay.

Both proposed methods can improve in several aspects, and here I would like to suggest the following future extensions to our work.

All proposed PPR methods for network coding, including F-PRNC and Fly-PRAC, assume that the coefficient vector generators are either synchronized between the sender and receiver nodes or the coefficient vectors are received without error if conveyed within packets. Further, PPR schemes also

assume that checksums in packets are also error-free. In other words, any error in the checksums or coefficient vectors will interrupt the recovery process. An improved PPR scheme should recover and handle errors of all parts of a packet, including associated coefficients and checksum. In addition, the proposed methods are only considering unicast communications. Another improvement would be assessing the performance of PPR schemes for multicast scenarios.

Bibliography

[1] J. Cabrera, G. Nguyen, D. E. Lucani, M. Pedersen, and F. H. P. Fitzek. Taking the trash back in: Practical joint channel and network coding for improving IEEE 802.11 networks. In *Eur. Wireless 2017; 23th Eur. Wireless Conf.*, pages 1–5, May 2017.

[2] Nicole Todtenberg and Rolf Kraemer. A survey on bluetooth multi-hop networks. *Ad Hoc Networks*, 93:101922, 2019.

[3] Drew Gislason. *Zigbee wireless networking*. Newnes, 2008.

[4] Kaveh Pahlavan and Prashant Krishnamurthy. Evolution and impact of wi-fi technology and applications: a historical perspective. *International Journal of Wireless Information Networks*, 28(1):3–19, 2021.

[5] Kalvin Bahia and Anne Delaporte. *The State of Mobile Internet Connectivity 2020*. GSMA Intelligence, 2020.

[6] Franck Muteba, Karim Djouani, and Thomas Olwal. A comparative survey study on lpwa iot technologies: Design, considerations, challenges and solutions. *Procedia Computer Science*, 155: 636–641, 2019. The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2019).

[7] A Forouzan Behrouz and CF Sophia. Data communications and networking. *Forouzan with Sophia Chung Fegan*, 2007.

[8] David Eckhardt and Peter Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. *SIGCOMM Comput. Commun. Rev.*, 26(4):243–254, August 1996.

[9] Sheryl L Howard, Christian Schlegel, Kris Iniewski, and Kris Iniewski. Error control coding in low-power wireless sensor networks: When is ecc energy-efficient? *EURASIP Journal on Wireless Communications and Networking*, 2006:1–14, 2006.

[10] Z. Hajjarian Kashani. Power optimised channel coding in wireless sensor networks using low-density parity-check codes. *IET Communications*, 1:1256–1262(6), December 2007.

[11] Rudolf Ahlswede, Ning Cai, S-YR Li, and Raymond W Yeung. Network information flow. *IEEE Trans. Inf. Theory*, 46(4):1204–1216, July 2000.

[12] Majid Ghaderi, Don Towsley, and Jim Kurose. Network coding performance for reliable multicast. In *MILCOM 2007 - IEEE Military Communications Conference*, pages 1–7, 2007.

[13] M. Ghaderi, D. Towsley, and J. Kurose. Reliability gain of network coding in lossy wireless networks. In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pages 2171–2179, 2008.

[14] Riccardo Bassoli, Hugo Marques, Jonathan Rodriguez, Kenneth W Shum, and Rahim Tafazolli. Network coding theory: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1950–1978, 2013.

[15] Peyman Pahlevani, Martin Hundebøll, Morten Pedersen, Daniel Lucani, Hassan Charaf, Frank P Fitzek, Hamidreza Bagheri, and Marcos Katz. Novel concepts for device-to-device communication using network coding. *IEEE Communications Magazine*, 52(4):32–39, 2014.

[16] Peyman Pahlevani, Achuthan Paramanathan, Martin Hundebøll, Janus Heide, Stephan A Rein, and Frank HP Fitzek. Reliable communication in wireless meshed networks using network coding. In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5. IEEE, 2012.

[17] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *IEEE Int. Symp. Inf. Theory, 2003. Proceedings.*, page 442, June 2003.

[18] Parastoo Sadeghi, Ramtin Shams, and Danail Traskov. An optimal adaptive network coding scheme for minimizing decoding delay in broadcast erasure channels. *EURASIP Journal on Wireless Communications and Networking*, 2010:1–14, 2010.

[19] Janus Heide, Morten V Pedersen, Frank HP Fitzek, and Muriel Médard. On code parameters and coding vector representation for practical rlnc. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5. IEEE, 2011.

[20] Danilo Silva, Weifei Zeng, and Frank R. Kschischang. Sparse network coding with overlapping classes. In *2009 Workshop on Network Coding, Theory, and Applications*, pages 74–79, 2009.

[21] Suzie Brown, Oliver Johnson, and Andrea Tassi. Reliability of broadcast communications under sparse random linear network coding. *IEEE Transactions on Vehicular Technology*, 67(5):4677–4682, 2018.

[22] Pablo Garrido, Daniel E Lucani, and Ramón Agüero. Markov chain model for the decoding probability of sparse network coding. *IEEE Transactions on Communications*, 65(4):1675–1685, 2017.

[23] Juan A. Cabrera, Raphael Steppert, Frank Gabriel, and Frank H. P. Fitzek. Do not waste the waste: Packetized rateless algebraic consistency for ieee 802.11 networks. In *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2021.

[24] Jin Xie, Wei Hu, and Zhenghao Zhang. Revisiting partial packet recovery in 802.11 wireless lans. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, page 281–292, New York, NY, USA, 2011. Association for Computing Machinery.

[25] G. Angelopoulos, M. Médard, and A. P. Chandrakasan. Harnessing partial packets in wireless networks: Throughput and energy benefits. *IEEE Trans. Wireless Commun.*, 16(2):694–704, Nov. 2017.

[26] H. K. Nazari, K. Ghassabi, P. Pahlevani, and D. E. Lucani. Improving the decoding speed of packet recovery in network coding. *IEEE Commun. Lett.*, 25(2):351–355, Feb. 2021.

[27] Kurniawan D Irianto, Juan A Cabrera, Giang T Nguyen, Hani Salah, and Frank HP Fitzek. S-PRAC: Fast partial packet recovery with network coding in very noisy wireless channels. In *2019 Wireless Days (WD)*, pages 1–7, Manchester, United Kingdom, Apr. 2019.

[28] Suelen Laurindo, Ricardo Moraes, Carlos Montez, and Francisco Vasques. Combining network coding and retransmission techniques to improve the communication reliability of wireless sensor network. *Information*, 12(5), 2021. ISSN 2078-2489.

[29] Sang-Woon Jeon, Sang Won Choi, Juyeop Kim, and Won-Yong Shin. Cellular-aided device-to-device communication: The benefit of physical layer network coding. *IEEE Communications Letters*, 20(11):2324–2327, 2016.

[30] Sachin Katti, Shyamnath Gollakota, and Dina Katabi. Embracing wireless interference: Analog network coding. *SIGCOMM Comput. Commun. Rev.*, 37(4):397–408, August 2007. ISSN 0146-4833.

[31] Pouya Ostovari, Jie Wu, and Abdallah Khreishah. *Network Coding Techniques for Wireless and Sensor Networks*, pages 129–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[32] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[33] Prasan Kumar Sahoo, Sudhir Ranjan Pattanaik, and Shih-Lin Wu. Design and analysis of a low latency deterministic network mac for wireless sensor networks. *Sensors*, 17(10), 2017. ISSN 1424-8220.

[34] L. Qi, Wenyu Sun, and Yiju Wang. Numerical multilinear algebra and its applications. *Frontiers of Mathematics in China*, 2:501–526, 2007.

[35] A. Zarei, P. Pahlevani, and D. E. Lucani. An analytical model for sparse network codes: Field size considerations. *IEEE Commun. Lett.*, 24(4):729–733, Apr. 2020.

[36] P. Sindhu. Retransmission error control with memory. *IEEE Trans. Commun.*, 25(5):473–479, May 1977.

[37] Grace R. Woo, Pouya Kheradpour, Dawei Shen, and Dina Katabi. Beyond the bits: Cooperative packet recovery using physical layer information. In *Proc. 13th Ann. ACM Int. Conf. Mobile Comput. and Netw.*, pages 147–158, Montréal, Québec, Canada, Sept. 2007.

[38] Kyle Jamieson and Hari Balakrishnan. Ppr: Partial packet recovery for wireless networks. In *Proc. 2007 Conf. Appl., Technol., Architectures, and Protocols for Comput. Commun.*, pages 409–420, Kyoto, Japan, Aug. 2007.

[39] Raghu K. Ganti, Praveen Jayachandran, Haiyun Luo, and Tarek F. Abdelzaher. Datalink streaming in wireless sensor networks. In *Proc. 4th Int. Conf. Embedded Networked Sensor Syst.*, pages 209–222, Boulder, Colorado, USA, Oct. 2006.

[40] A. P. Iyer, G. Deshpande, E. Rozner, A. Bhartia, and Lili Qiu. Fast resilient jumbo frames in wireless LANs. In *2009 17th Int. Workshop Qual. of Service*, pages 1–9, Charleston, SC, USA, July 2009.

[41] Kurniawan D. Irianto, Giang T. Nguyen, Hani Salah, and Frank H.P. Fitzek. Partial packet in wireless networks: a review of error recovery approaches. *IET Communications*, 14(2):186–192, 2020.

[42] M. S. Mohammadi, Q. Zhang, and E. Dutkiewicz. Exploiting partial packets in random linear codes using sparse error recovery. In *2015 IEEE Int. Conf. Commun. (ICC)*, pages 2577–2582, London, UK, June 2015.

[43] M. S. Mohammadi, Q. Zhang, and E. Dutkiewicz. Reading damaged scripts: Partial packet recovery based on compressive sensing for efficient random linear coded transmission. *IEEE Trans. Commun.*, 64(8):3296–3310, June 2016.

[44] Tracey Ho and Desmond Lun. *Network coding: an introduction*. Cambridge University Press, 2008.

[45] Sriram Pemmaraju and Steven Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press, 2003.

[46] S. Leung-Yan-Cheong, E. Barnes, and D. Friedman. On some properties of the undetected error probability of linear codes (corresp.). *IEEE Transactions on Information Theory*, 25(1): 110–112, 1979.

[47] Morten V. Pedersen, Janus Heide, and Frank H. P. Fitzek. Kodo: An open and research oriented network coding library. In Vicente Casares-Giner, Pietro Manzoni, and Ana Pont, editors, *Networking 2011 Workshops*, pages 145–152, Berlin, Heidelberg, 2011.

[48] "Global ping statistics - wondernetwork". URL `https://www.wondernetwork.com/pings`. Accessed on: Sept. 13, 2020.