# Improved Constructions of Coding Schemes for the Binary Deletion Channel and the Poisson Repeat Channel

Roni Con* Amir Shpilka*

**Abstract**

This work gives an explicit construction of a family of error correcting codes for the binary deletion channel and for the Poisson repeat channel. In the binary deletion channel with parameter $p$ ($\mathrm{BDC}_p$) every bit is deleted independently with probability $p$. A lower bound of $(1-p)/9$ is known on the capacity of the $\mathrm{BDC}_p$ [MD06], yet no explicit construction is known to achieve this rate. We give an explicit family of codes of rate $(1-p)/16$, for every $p$. This improves upon the work of Guruswami and Li [GL18] that gave a construction of rate $(1-p)/120$. The codes in our family have polynomial time encoding and decoding algorithms.

Another channel considered in this work is the Poisson repeat channel with parameter $\lambda$ ($\mathrm{PRC}_\lambda$) in which every bit is replaced with a discrete Poisson number of copies of that bit, where the number of copies has mean $\lambda$. We show that our construction works for this channel as well. As far as we know, this is the first explicit construction of an error correcting code for $\mathrm{PRC}_\lambda$.

*Department of Computer Science, Tel Aviv University, Tel Aviv, Israel. The research leading to these results has received funding from the Israel Science Foundation (grant number 552/16) and from the Len Blavatnik and the Blavatnik Family foundation.

Part of this paper has been published in the proceedings of the 2020 IEEE International Symposium on Information Theory (ISIT).

# Contents

## 1 Introduction

This work deals with constructing error correcting codes for the channels called the binary deletion channel (BDC for short) and the Poisson repeat channel (PRC for short).

Loosely speaking, a channel is a medium over which messages are sent. A channel is defined by the way in which it introduces errors to the transmitted messages (also called codewords when they come from an error correcting code). Before describing the channels that we consider in this work, we first discuss the two main error models - a worst case model and an average case model.

The first model, which is very common in the theory of computation and has found many applications there, is called the Hamming model [Ham50]. This is a worst case setting in which a transmitted message is subjected to an adversarial corruption of a fraction $p$ of its entries and we must recover the original message regardless of the location of the errors. Thus, if the adversary is allowed to corrupt a fraction $p$ of the entries of a transmitted message, then an error correcting code for this channel that allows perfect recovery is a subset of the messages such that any two codewords (i.e. elements of the code) have normalized hamming distance larger than $2p$. The second error model, which is the one relevant to our work, was first considered by Shannon in his pioneering work [Sha48]. This is an average case model in which a transmitted message is subjected to a random corruption such as bit flips, bit erasures, bit deletions, etc., where each bit is corrupted independently at random according to some distribution.[1] A channel is basically determined by the probability distribution of corruptions. Since the corruption is random, it can be the case that the whole word is corrupted. In particular, in this setting, the most we can expect from the decoder is to decode the original word with high probability (over the randomness of the corruptions).

The two most studied channels are the Binary Erasure Channel ($\mathrm{BEC}_p$) where each bit is independently replaced by a question mark with probability $p$ and the Binary Symmetric Channel ($\mathrm{BSC}_p$) where each bit is independently flipped with probability $p$.

In this work we consider the BDC with parameter $p$. This channel models the situation where bits of a transmitted message are deleted (i.e. removed) from the message randomly and independently with probability $p$. In particular, if a message of length $n$ was transmitted on the $\mathrm{BDC}_p$ then the length of the received message is concentrated around $(1-p)\cdot n$. We note that the output of the BDC is very different from that of the BEC or the BSC. For example, if we transmit the message 1110101 over each of the channels and corruptions occurred in locations 2 and 5, then the BEC will return the word 1?10?01, the BSC will return 1010001, and the BDC will return 11001. In particular, while the BEC and the BSC do not affect the length of messages transmitted over them, the BDC does exactly that. Thus, unlike the BEC and BSC, the BDC causes synchronization errors. In fact, one of the main reasons for introducing the BDC was to model synchronization errors in communication.

The motivation to study the BDC is obvious. It is not just a theoretical object as it describes a real-life scenario in which there is a loss of information that was sent on some physical layer as well as synchronization errors. Moreover, the surveys [Mit09, MBT10] indicate that tools that were developed in the context of the BDC are useful in the study of other questions. An example of such a question is the trace reconstruction problem,

[1]This description corresponds to a memoryless channel, which is the most common model.

which has applications in computational biology and DNA storage systems [BLC+16]. The problem that we study in this work is the construction of explicit error correcting codes of (relatively) high rate (we will soon explain this notion) for the $\mathrm{BDC}_p$.

Another model that we consider is the PRC that was first introduced in the work of Mitzenmacher and Drinea [MD06]. In the PRC with parameter $\lambda$, each bit of the message is (randomly and independently) replaced with a discrete number of copies of that bit, distributed according to the Poisson distribution with parameter $0 < \lambda$. In particular, with probability $e^{-\lambda}$ the bit is deleted from the message (i.e. this channel can cause synchronization errors similar to the BDC). This channel can model, for example, messages sent using a keyboard that has tendency to get stuck so a key cannot be pressed or can get stuck and then its symbol is repeated several times. While the PRC is less motivated by practical applications (we are unaware of any applications of this channel besides in the study of the BDC), it is closely related to the BDC as demonstrated in the work of Mitzenmacher and Drinea [MD06], Drinea and Mitzenmacher [DM07] and Cheraghchi [Che18]. In particular, the lower bound on the capacity (a notion that we explain shortly) of $\mathrm{BDC}_p$ of $(1-p)/9$ [MD06] relies on a reduction from the $\mathrm{PRC}_\lambda$. We too exploit the connection between the BDC and the PRC, and using our construction for the BDC we obtain explicit constructions of error correcting codes for the PRC.

To explain the question that we study we need some basic notions from coding theory. Recall that a binary[2] error correcting code can be described either as an encoding map $C : \{0,1\}^k \to \{0,1\}^n$ or, abusing notation, as the image of such a map $C$. The rate of such a code $C$ is $\mathrm{Rate}(C) = k/n$, which intuitively captures the amount of information encoded in every bit of a codeword. Naturally, we would like the rate to be as large as possible, but there is a tension between the rate of the code and the amount of errors/noise it can tolerate.

One of the most fundamental questions when studying a channel is to determine its capacity, i.e., the maximum achievable transmission rate over the channel that still allows recovering from the errors introduced by the channel, with high probability. Shannon proved in his seminal work [Sha48] that the capacity of the $\mathrm{BSC}_p$ is $1-h(p)$, where $h(\cdot)$ is the binary entropy function (for $0 < x < 1$, $h(x) = -x\log x - (1-x)\log 1-x$).[3] I.e., there are codes with block lengths going to infinity, whose rates converge to $1-h(p)$, that can recover with high probability from the errors inflicted by the channel. Elias [Eli55], who introduced the $\mathrm{BEC}_p$, proved that its capacity is $1-p$.

What about the capacity of the $\mathrm{BDC}_p$? In spite of many efforts (see [Mit09]), the capacity of the $\mathrm{BDC}_p$ is still not known and it is an outstanding open challenge to determine it. Yet, for the extremal cases, the asymptotic behavior is somewhat understood. In the regime where $p \to 0$ the capacity approaches to $1-h(p)$ [KMS10], i.e. it approaches the capacity of the $\mathrm{BSC}_p$. In [MD06], the authors showed that the capacity is at least $(1-p)/9$ for all $p \in (0,1)$. In particular, even if $p$ is extremely close to 1, there are codes of positive rate that allow reliable communication over this channel. Another somewhat surprising aspect of this result is that the asymptotic behavior is only a constant off from the capacity of the related $\mathrm{BEC}_p$. In the $\mathrm{BEC}_p$, we know how to build codes that nearly achieve its capacity of $1-p$ for every $p$. This is not the case for the $\mathrm{BDC}_p$, where the

[2]The case of codes over non-binary alphabets is very important of course, but in this work we only focus on binary codes.

[3]All logarithms in this paper are base 2.

best explicit construction known for the regime $p \to 1$, prior to this work, has rate of $(1-p)/120$ [GL18].

In this work we present and analyze a polynomial time construction of a family of codes for the $\mathrm{BDC}_p$ that achieves rate higher than $(1-p)/16$ for every $p$ and have polynomial time encoding and decoding algorithms. We also show that this construction yields a family of codes for $\mathrm{PRC}_\lambda$ of rate $\mathcal{R} > \lambda/17$ for $\lambda \leq 0.5$. This further emphasizes that these channels have much in common.

## 1.1 Previous Work

Much of the major results on the capacity of deletion type channels can be found in the excellent surveys of Mitzenmacher's and Mercier et al. [Mit09, MBT10]. We highlight some of the results related to the regime where $p$ tends to 1 as this regime is the focus of this paper.

The best known lower bound on the capacity is due to Mitzenmacher and Drinea [MD06] that showed a lower bound of $(1-p)/9$ for all $p$, meaning that there are codes of this rate such that every transmitted codeword is decoded correctly with high probability. Their proof is existential and does not yield an explicit construction with this rate. As far as we know there is no explicit construction that achieves rate of $(1-p)/9$. A more recent work by Guruswami and Li [GL18] presents a deterministic code construction for the $\mathrm{BDC}_p$ with rate $(1-p)/120$ for all values of $p$. This rate is smaller then Mitzenmacher's bound, but it is the first construction with rate that scales proportionally to $(1-p)$ for $p \to 1$. In [Dal11] an upper bound of $0.4143(1-p)$ was shown on the capacity of $\mathrm{BDC}_p$ for $p \to 1$, meaning that there are no error correcting codes that achieve this rate for the $\mathrm{BDC}_p$.[4]

Deletion correction is studied also in the adversarial model, i.e., when there is an adversary that can delete up to some threshold number of symbols. In fact, works dealing with the adversarial model considered the more general case in which the adversary is also allowed, in addition to deletions, to insert symbols, i.e., to add a new symbol from the alphabet between two adjacent symbols in the codeword. In this context of adversarial deletions and insertions, the work of Haeupler and Shahrasbi [HS17] gave efficient insertion-deletion (insdel for short) codes over large alphabet, which are almost optimal in rate-distance trade-off. In particular, for every $\epsilon > 0$ and $\delta \in (0,1)$ there is a code $C$ with rate $1-\delta-\epsilon$ that can efficiently correct a $\delta$ fraction of insertions and deletions and its alphabet size is given by $|\Sigma| = O_\epsilon(1)$. We note that this construction does not give a binary code. In the high rate regime, Guruswami and Wang [GW17] showed that there are binary codes of rate $1-\tilde{O}(\sqrt{\delta})$ that can correct $\delta n$ worst-case deletions in polynomial time.

## 1.2 Our Results

In this work, we improve the construction presented in [GL18] and construct an explicit family of codes for the binary deletion channel with rate at least $(1-p)/16$ for any

[4]We note that Dalai's proof was computer assisted. A recent work by Cheraghchi [Che18] gave an upper bound on the capacity of the $\mathrm{BDC}_p$ for $p \geq 1/2$ of $(1-p)\log((1+\sqrt{5})/2)$ without computer assistance.

$p \in (0,1)$ that have polynomial time encoding and decoding algorithms.

**Theorem 1.1.** *Let $p \in (0,1)$. There exist a family of binary error correcting codes $\{C_i\}_{i=1}^{\infty}$ for the BDC$_p$ where the block length of $C_i$ goes to infinity as $i \to \infty$ and*

1. *$C_i$ can be constructed in time polynomial in its block length.*
2. *$C_i$ has rate at least $(1-p)/16$.*
3. *$C_i$ is decodable in quadratic time and encodable in linear time.*

As mentioned earlier, we show that the same construction works for the PRC$_\lambda$ as well. In particular we prove,

**Theorem 1.2.** *Let $\lambda \leq 0.5$. There exist a family of binary error correcting codes $\{C_i\}_{i=1}^{\infty}$ for PRC$_\lambda$ where the block length of $C_i$ goes to infinity as $i \to \infty$ and*

1. *$C_i$ can be constructed in time polynomial in its block length.*
2. *$C_i$ has rate $\mathcal{R}_i > \lambda/17$.*
3. *$C_i$ is decodable in quadratic time and encodable in linear time.*

To the best of our knowledge, this is the first explicit construction of an error correcting code for the PRC$_\lambda$.

## 1.3 Construction and Proof Overview

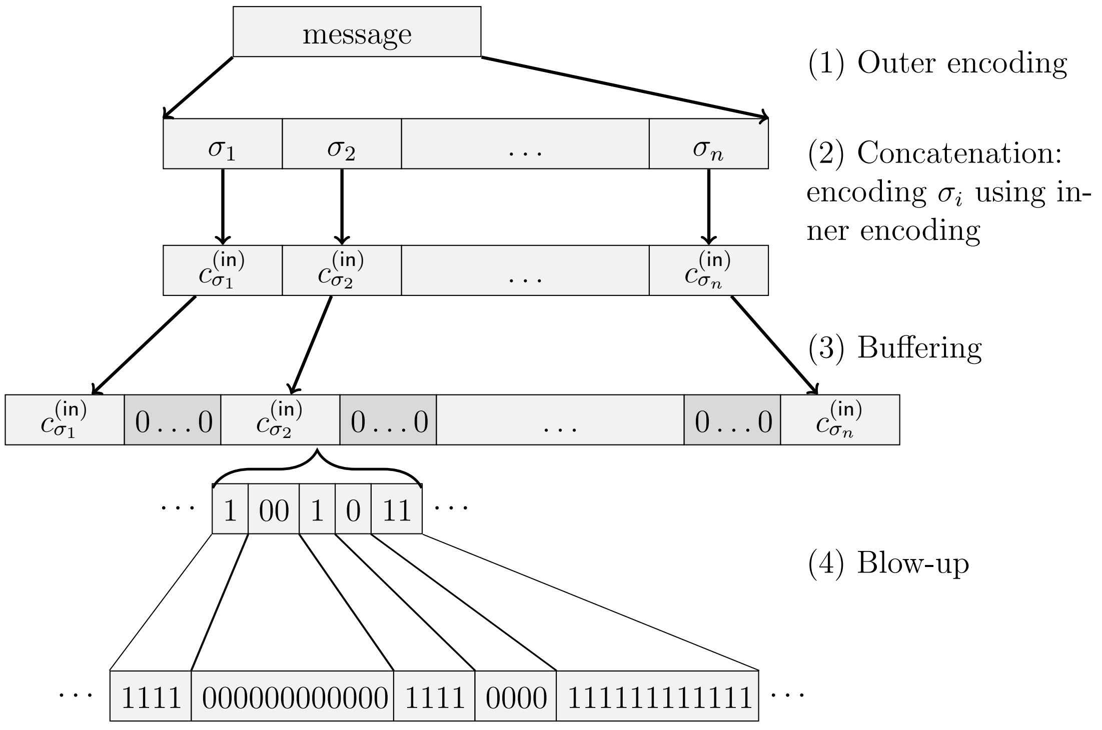
Our construction follows the footsteps of the construction of Guruswami and Li [GL18] with some important modifications. We next describe the construction and then its analysis.

**Construction:** There are several layers to our construction as depicted in Figure 1 on page 7. The first two layers come from code concatenation while the third and fourth layers blow-up the code further by repeating symbols and inserting "buffers" between inner codewords. These four layers are similar to those in the construction of [GL18] and the main difference between the constructions is that we use a different inner code and the blow-up in our construction is considerably smaller. We now describe each step in more detail.

Recall that code concatenation is the operation of viewing the message as a shorter message over a larger alphabet, then applying an error correcting code over the large alphabet (the outer code) to the message and, finally, viewing each symbol of the encoded message as a short message over $\{0,1\}$, it is encoded using a binary error correcting code (the inner code).

In our construction, we view the messages as strings of length $k$ over the alphabet $\Sigma = \{0,1\}^{m'}$ (where $m'$ is some constant that we later optimize). As an outer code, we use the code from [HS17], which is an efficient insertion-deletion code with rate close to 1 over $\Sigma$. This code returns a word $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \Sigma^n$.

We construct our inner code using a greedy algorithm. First we consider all binary strings of length $m$ which consist of exactly $\beta_1 m$ 1-runs and $0.5(1-\beta_1)m$ 2-runs (i.e.



message
(1) Outer encoding
$\sigma_1$
$\sigma_2$
...
$\sigma_n$
(2) Concatenation: encoding $\sigma_i$ using inner encoding
$c^{(\mathsf{in})}_{\sigma_1}$
$c^{(\mathsf{in})}_{\sigma_2}$
...
$c^{(\mathsf{in})}_{\sigma_n}$
(3) Buffering
$c^{(\mathsf{in})}_{\sigma_1}$
0...0
$c^{(\mathsf{in})}_{\sigma_2}$
0...0
...
0...0
$c^{(\mathsf{in})}_{\sigma_n}$
··· 1 00 1 0 11 ···
(4) Blow-up
··· 1111 000000000000 1111 0000 111111111111 ···

Figure 1: The encoding process.

alternating blocks of 0's and 1's where each block length is $\leq 2$) where $\beta_1$ is a parameter that we will optimize later. Then, we add a codeword to our codebook if it does not contain a subsequence of length $\geq m - \delta m$ that is also a subsequence of any codeword that is already in our codebook. Note that even though the construction time is exponential in $m$, as $m = O(1)$ in our construction this does not affect the run time by more than a constant factor.

The encoding process is thus as follows (see also Figure 1). We first encode the message using the outer code to a codeword of length $n$ over $\Sigma$. Then, the concatenation process takes every symbol, $\sigma_i$ of the outer codeword and maps it to a codeword from the inner code, i.e., a concatenated codeword is of the form $c_1 \circ c_2 \circ \cdots \circ c_n$ where $c_i = \mathrm{ENC}_{\mathrm{in}}(\sigma_i)$, where $\mathrm{ENC}_{\mathrm{in}}$ is the encoding function of the inner code. This is not the end of the story. In order for the concatenated code to overcome a large amount of deletions caused by the channel we add an additional layer of encoding:

1. We place long buffers of zeros (of length $N_B$) between inner codewords. This step helps the decoder identify where an inner codeword starts and where it ends.

2. We replace each 1-run with an $N_1$-run and each 2-run will become an $N_2$-run (runs of length $N_1$ and $N_2$). This helps the decoder identify if the run in the inner codeword was a run of length 1 or 2.

This step is also similar to the construction of [GL18], however, perhaps surprisingly, since we restrict our inner codewords to have a fixed number of 1-runs and 2-runs, this enables us to have $N_1$ and $N_2$ considerably smaller than the blow-up parameter used in [GL18]. It is clear that the code construction is efficient as the outer code of [HS17] can be encoded efficiently and the inner code is of constant length and thus can also be encoded efficiently. The last step is clearly efficient (as $B, N_1$ and $N_2$ are constants).

**Decoding:** We now describe our decoding algorithm. First, we identify the buffers in order to divide the string into "decoding windows" that should ideally represent corrupted inner codewords. Second, every decoding window is decoded in the following way: Every run longer than some threshold $T$ is replaced with a 2-run (of the same symbol) and every run of length $\leq T$ is replaced with a 1-run. The third step of the decoding is to use a brute force decoding algorithm on each decoded window to find the closest inner codeword. Since the inner code's block length is constant this step takes constant time for every such window and hence runs in linear time in the length of the word. The last step in the decoding algorithm is to run the decoding algorithm of the outer code as given in [HS17]. This algorithm runs in time quadratic in the outer code's block length. Hence the total run time of our decoder is quadratic in the length of the message.

**Analysis:** Our analysis classifies errors to three types:

1. Buffer deletions: these are deletions that caused a buffer between inner codewords to completely disappear.
2. Spurious buffers: these are deletions of many 1's that caused the algorithm to mistakenly identify a buffer inside an inner codeword.
3. Wrong decoding of inner codewords: these occur when the algorithm fails to decode correctly a corrupted inner codeword.

The first and second error types can happen in the first stage of the algorithm, i.e., when the decoder identifies the buffers between blown-up inner codewords. First, the decoder might not identify a buffer when a large portion of the buffer was deleted and second, the decoder might mistakenly think that there is a buffer inside an inner codeword if many consecutive runs of the symbol 1 were deleted. We show by using simple concentration bounds that both error types happen with exponentially small probability in $m$, the inner code block length (as $m = O(1)$ this is a constant probability, but it is still small enough to allow our construction to work). The third error type we consider is when the edit distance between the sent inner codeword and the corresponding string obtained from the second step of the decoding algorithm is greater than $\delta_{\text{in}}m$, the inner code's decoding radius. In this case, the decoding algorithm of the inner code might output a wrong codeword. While this can happen, we show that the *expected* edit distance between the original inner codeword and the decoded inner codeword[5] is smaller than $\delta_{\text{in}}m$, for a large enough $m$, and furthermore, the edit distance is concentrated around its mean. Hence, we expect to decode successfully most of the inner codewords. Finally, we show that this reasoning implies that the decoding algorithm of the outer code, which is executed at the last step of our decoding algorithm, succeeds with probability $1 - \exp(-\Omega(n))$.

[5]In the proof we use the term decoded window as we are never really sure when a codeword started and ended, but this does not affect the intuition.

In terms of complexity, we show that even though the construction and decoding of the inner code are exponential in the inner code's block length, the overall complexity (construction, encoding, and decoding) is dominated by the complexity of the outer code which has polynomial time encoding and decoding algorithms thanks to [HS17].

**Comparison to [GL18].** We end this high-level summary by elaborating more on the main similarities and differences between our construction and the construction of Guruswami and Li [GL18]. Our scheme, as well as our decoding algorithm, follow closely the scheme and algorithm of Guruswami and Li. In particular, at a high level, the encoding layers are the same as in [GL18], meaning that both constructions use concatenation with the outer code from [HS17], place long buffers between inner codewords and blow-up the code. Since the encoding layers are similar, the decoding steps in both papers are also similar: first identify the buffers, then use a threshold to distinguish between 1-runs and 2-runs, then use brute force to decode the inner codewords, and finally use the decoder of [HS17]. The main differences between our scheme and the scheme from [GL18] are in the inner code that is used, the blow-up process which is finer in our scheme and our analysis which is more fine-tuned:

- The inner code that was used in [GL18] has the property that every codeword consists of 1-runs and 2-runs, but they do not have restriction on the number of 1-runs and 2-runs. In contrast, in our work, all inner codewords have the same number of 1-runs and 2-runs. This property allows us to increase the rate of the inner code compared to [GL18] (See Propositions 3.4 and 3.5 and the discussion following them), while maintaining its robustness against insertions and deletions.
- In [GL18] the authors blow-up the code by replacing every bit with $60/(1-p)$ copies of that bit. Instead of blowing-up every single bit, we blow-up each 1-run to an $N_1$-run and each 2-run to an $N_2$-run where $N_1 \neq N_2$ and both are significantly smaller than $60/(1-p)$ ($N_1 \approx 6/(1-p)$ for example). Thus, the effect of the blow-up on the rate of our code is significantly smaller than in [GL18].
- We improve on the analysis in [GL18], of the edit distance between decoded inner codewords and the original inner codewords, by better accounting the effect of decoding errors on the edit distance. One more improvement lies in our analysis where instead of using the Chernoff bound to upper bound the probability of certain events, we use the fact that binomial distributions with fixed expectations converge to a Poisson distribution. This gives a better upper bound which eventually leads to some saving when optimizing parameters. Our analysis further highlights the tight connection between the BDC and the PRC via the convergence of the binomial distribution to the Poisson distribution.

These modifications, as well as a careful choice of parameters, is the reason for the great saving in the rate compared to [GL18].

## 1.4 Organization

The paper is organized as follows. In Section 2 we introduce the basic notation as well as some well known facts from probability and from previous papers. Section 3 contains the construction of our inner code. In Section 4 we give our construction and in Section 5 we give its analysis. We give slightly improved bounds for fixed values of $p$ in Section 6. Finally, Section 7 explains how to carry our construction and analysis to the PRC.

## 2 Preliminaries

For an integer $k$, we denote $[k] = \{1, 2, \ldots, k\}$. Throughout this paper, $\log(x)$ refers to the base-2 logarithm and $h(x)$ denotes the binary entropy function, that is, $h(x) = -x\log(x) - (1-x)\log(1-x)$, for $0 < x < 1$. We use $\Sigma$ to denote an alphabet and $\Sigma^*$ to denote all the finite length strings over $\Sigma$. For $s \in \Sigma^*$ we denote by $|s|$ the length of $s$.

**Definition 2.1.** *Let $s \in \Sigma^*$. The operation in which we remove a symbol from $s$ is called a* deletion *and the operation in which we place a new symbol from $\Sigma$ between two consecutive symbols in $s$ is called an* insertion*.*

*A* substring *of $s$ is a string obtained by taking consecutive symbols from $s$. A* subsequence *of $s$ is a string obtained by removing some (possibly none) of the symbols in $s$.*

**Definition 2.2.** *Let $s, s' \in \Sigma^*$. A* longest common subsequences *between $s$ and $s'$, is a subsequence $s_{\text{sub}}$ of both $s$ and $s'$, of maximal length. We denote by $|\text{LCS}(s, s')|$ the length of a longest common subsequence.*[6]

*The* edit distance *between $s$ and $s'$, denoted by $ED(s, s')$, is the minimal number of insertions and deletions needed in order to turn $s$ into $s'$.*

**Lemma 2.3** (See e.g. Lemma 12.1 in [CR03])**.** *It holds that* $\text{ED}(s, s') = |s| + |s'| - 2\,|\text{LCS}(s, s')|$.

**Definition 2.4.** *Let $s$ be a string. A* run *$r$ in $s$ is a single-symbol substring of $s$ such that the symbol before the run and the symbol after the run are different from the symbol of the run. A run of length $\ell$ will be denoted as $\ell$-run.*

For example, consider the string $\langle 0111001 \rangle$. It can be written as the (string) concatenation of the alternating runs $0 \circ 111 \circ 00 \circ 1$. Clearly, every binary string is a concatenation of runs of alternating symbols. The following lemma of Levenshtein will be useful in the analysis of the rate of our inner code.

**Lemma 2.5.** *[Lev66] Let $s$ be a string and let $r(s)$ be the number of runs in $s$. There are at most*

$$\binom{r(s) + d - 1}{d}$$

*different subsequences of $s$ of length $|s| - d$.*

[6]Note that a longest common subsequence may not be unique as there can be a number of subsequences of maximal length.

## 2.1 Facts from Probability

We use two probability distributions in this paper. The *binomial distribution* with parameters $n$ and $p$, denoted $\mathrm{Bin}(n,p)$, is the discrete probability distribution of the number of successes in a sequence of $n$ independent trials, where the probability of success in each trial is $p$ and the probability of failure is $1-p$. The second distribution is the *discrete Poisson distribution* with parameter $\lambda$, denoted as $\mathrm{Poisson}(\lambda)$ which is defined with the following probability mass function

$$\Pr\left[X=k\right]=\frac{e^{-\lambda}\lambda^k}{k!}.$$

A well known fact about Poisson distribution is

**Lemma 2.6.** *[MU05, Lemma 5.2] Let $X$ and $Y$ be two independent Poisson random variables with parameters $\mu_1$ and $\mu_2$. I.e., $X\sim\mathrm{Poisson}(\mu_1)$ and $Y\sim\mathrm{Poisson}(\mu_2)$. Then $Z=X+Y$ is a Poisson random variable with parameter $\mu_1+\mu_2$.*

We shall use the following simple lemma in our analysis:

**Lemma 2.7.** *Fix $T$ to be a non negative integer and let $Y(\lambda)\sim\mathrm{Poisson}(\lambda)$. Then the function*

$$f(\lambda):=\Pr[Y(\lambda)\le T]=e^{-\lambda}\sum_{i=0}^{T}\frac{\lambda^i}{i!}$$

*is monotonically decreasing in $\lambda$.*

*Proof.* It holds that

$$\frac{\mathrm{d}f}{\mathrm{d}\lambda}(\lambda)=-e^{-\lambda}\sum_{i=0}^{T}\frac{\lambda^i}{i!}+e^{-\lambda}\sum_{i=0}^{T-1}\frac{\lambda^i}{i!}=-e^{-\lambda}\frac{\lambda^T}{T!}<0\ .$$

□

The next theorem shows that if we let $n$ tend to infinity and $p$ tend to zero under the restriction that $p\cdot n=\lambda$, then the binomial distribution converges to the Poisson distribution with parameter $\lambda$.

**Theorem 2.8.** *[MU05, Theorem 5.5] Let $\lambda>0$ be fixed. Let $\{X_n\}$ be a sequence of binomial random variables such that $X_n\sim\mathrm{Bin}(n,p)$, and $\lim_{n\to\infty}np=\lambda$. Then, for any fixed $k$,*

$$\lim_{n\to\infty}\Pr[X_n=k]=\frac{e^{-\lambda}\lambda^k}{k!}\ .$$

The next theorem provides more information about the binomial distribution in the regime where $np=\lambda$. Specifically, it tells us when is $\Pr[X\le T]$ an increasing function of $n$.

**Theorem 2.9.** *[AS65, Theorem 2.1] Let $\{X_n\}$ be a sequence of binomial random variables with parameters $n$ and $p=\lambda/n$. Let $T$ be some parameter. Set $f(n):=\Pr[X_n\le T]$.*

1. *If $T \leq \lambda - 1$ then for every $n \geq \lambda$, $f(n)$ is monotonically increasing in $n$.*
2. *If $\lambda \leq T$ then for every $n \geq T$, $f(n)$ is monotonically decreasing in $n$.*

For concentration bounds, we will use the following versions of the Chernoff bounds.

**Lemma 2.10.** *[MU05, Theorems 4.4 and 4.5] Suppose $X_1, \ldots, X_n$ are independent identically distributed random variables taking values in $\{0,1\}$. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X_i]$. Then, for any $0 < \alpha < 1$:*

$$\Pr[X \geq (1+\alpha)n\mu] \leq e^{-\frac{\mu n \alpha^2}{3}}$$

*and*

$$\Pr[X \leq (1-\alpha)n\mu] \leq e^{-\frac{\mu n \alpha^2}{2}} .$$

When we have a Poisson random variable we shall use the following Chernoff bound

**Lemma 2.11.** *[MU05, Theorem 5.4] Let $X$ be a Poisson random variable with parameter $\mu$.*

1. *If $x > \mu$, then*
$$\Pr(X \geq x) \leq \frac{e^{-\mu}(e\mu)^x}{x^x} .$$
2. *If $x < \mu$,*
$$\Pr(X \leq x) \leq \frac{e^{-\mu}(e\mu)^x}{x^x} .$$

Another concentration bound we use is Hoeffding's inequality

**Theorem 2.12.** *[Hoe94, Theorem 2] If $X_1, X_2, \ldots, X_n$ are independent random variables with finite first and second moment and $a_i \leq X_i \leq b_i$ for $1 \leq i \leq n$. Let $X = \sum_{i=1}^n X_n$ and $\mu = \mathbb{E}[X]$ then for $t > 0$*

$$\Pr[X - \mu > t] < \exp\left(-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right) .$$

To approximate binomial coefficients we shall use the following lemma

**Lemma 2.13.** *For any $n, k \in \mathbb{N}$ such that $k/n \leq 1/2$ we have,*

$$2^{nh\left(\frac{k}{n}\right) - O(\log n)} \leq \binom{n}{k} \leq 2^{nh\left(\frac{k}{n}\right)} .$$

The proofs of the bounds follow from Stirling's formula, e.g., see [GRS12, Section 3.2]

## 2.2 The Code of Haeupler and Shahrasbi [HS17]

Our construction relies on the following code of Haeupler and Shahrasbi [HS17].

**Theorem 2.14** ([HS17, Theorem 1.1])**.** *For every $\epsilon_{\text{out}} > 0$ and $\delta_{\text{out}} \in (0,1)$ there exists $n_0$ so that for every $n > n_0$ there is an integer $k$ satisfying $k/n > 1 - \delta_{\text{out}} - \epsilon_{\text{out}}$, an alphabet $\Sigma$ of size $O_{\epsilon_{\text{out}}}(1)$ and encoding and decoding maps $E : \Sigma^k \mapsto \Sigma^n$, $D : \Sigma^* \mapsto \Sigma^k$, respectively, such that if $\mathrm{ED}(E(x), y) \leq \delta_{\text{out}} n$ then $D(y) = x$. Further $E$ and $D$ are explicit and can be computed in linear and quadratic time in $n$, respectively.*

We shall denote with $\mathcal{R}_{\text{out}} := k/n$ the rate of this code, which will be used as the outer code in our construction.

## 3 The Inner Code

In this section we describe the construction of our inner code. Before giving the construction we define a set of strings from which we shall pick our codewords.

**Definition 3.1.** *We denote with $S \subset \{0,1\}^*$ the set containing all binary strings $s$ that start and end with the symbol 1 and that contain only 1-runs and 2-runs.*

*Let $\beta_1 \in [0,1]$. Define $S_{m,\beta_1} \subset S$ to be the set of all $s \in S$ of length $m$, such that the number of 1-runs in $s$ is exactly $\beta_1 m$ and the number of 2-runs in $s$ is exactly $\beta_2 m = (1-\beta_1)m/2$. Denote $\beta := \beta_1 + \beta_2$.*

**Remark 3.2.** *We observe that as every string in $S$ begins and ends with the same symbol, the number of runs in it is odd. Accordingly, $\beta m$, in the definition of $S_{m,\beta_1}$, is an odd integer.*

Our goal in this section is to construct a code $C \subset S_{m,\beta_1}$ such that the length of a longest common subsequence of any two different codewords is $< m - \delta m$.

**Remark 3.3.** *Observe that the deletion channel is likely to output a word that is not in $S$. However, Algorithm 2 (given in Section 5), for decoding the final code, contains a threshold decoding step (Step 2 in Algorithm 2) that always returns a binary string in $S$. We further restrict our attention to $S_{m,\beta_1}$ as it somewhat simplifies the analysis.*

We construct this code using the natural greedy algorithm: We consider all strings in $S_{m,\beta_1} \subset S$ and greedily choose strings that are far from each other. To reason about the parameters of the code, we need the following propositions.

The first proposition gives an upper bound on the size of the "deletion ball", i.e., given a string $s \in S_{m,\beta_1}$ it upper bounds the number of different subsequences of $s$ of length $m - \delta m$, that belong to $S$.

**Proposition 3.4.** *Let $s \in S_{m,\beta_1}$. It holds that*

$$\#\{s' \in S \mid s' \text{ is a subsequence of } s \text{ and } |s'| = m - \delta m\} \le \binom{(\beta+\delta)m}{\delta m}.$$

*Proof.* By definition, $s$ is a binary string that contains exactly $\beta m$ runs. Let $H$ be the set of all the subsequences obtained from $s$ by applying $\delta m$ deletions. According to Lemma 2.5, the size of $H$ is at most $\binom{\beta m + \delta m - 1}{\delta m} < \binom{\beta m + \delta m}{\delta m}$. Clearly if we restrict further and consider only those strings in $H \cap S$ we can only decrease the size of the set. □

The second proposition upper bounds the size of the "insertion ball", i.e., given a string $s' \in S$ of length $m - \delta m$, it gives an upper bound on the number of strings $s \in S_{m,\beta_1}$ that can be obtained from $s'$ by performing $\delta m$ insertions. The proof of this proposition is considerably more elaborate.

**Proposition 3.5.** *Let $0 < \delta < \beta_1/3$. Fix $s_{\text{sub}} \in S$ such that $|s_{\text{sub}}| = m - \delta m$. The number of binary strings in $S_{m,\beta_1}$ that contain $s_{\text{sub}}$ as a subsequence is at most $\binom{\beta m}{\delta m}$.*

We note that the equivalent propositions from [GL18] gave upper bounds of $\binom{m}{\delta m}$ and $O(\delta m) \cdot \binom{m}{\delta m}$ respectively. The main reason for our saving is that we restrict our codewords to have exactly $\beta_1 m$ 1-runs and $\beta_2 m$ 2-runs. This saving is one of the places where we improve upon [GL18]. This improvement affects the rate of the inner code as we shall later see.

The proof of the proposition relies on an algorithm for generating all strings $s \in S_{m,\beta_1}$ such that $s_{\text{sub}}$ is a subsequence of $s$. As in [GW17, Lemma 2.3], in order to avoid over counting, we will generate all such $s$ by finding the lexicography first occurrence of $s_{\text{sub}}$ in $s$. We first explain the idea behind the algorithm and then prove Proposition 3.5.

Denote $s_{\text{sub}} = \langle b_1 b_2 \dots b_{m-\delta m} \rangle$, where $b_i \in \{0, 1\}$. In order to obtain a string $s \in S_{m,\beta_1}$ from $s_{\text{sub}}$ we need to choose indices $1 \leq n_1 < n_2 < \dots < n_{m-\delta m} \leq m$ for the bits of $s_{\text{sub}}$ in $s$. Moreover, to make sure that the locations chosen are indeed the lexicography first occurrence of $s_{\text{sub}}$ in $s$, the entries between $n_i$ and $n_{i+1}$ (for $1 \leq i \leq m - \delta m - 1$) must contain the opposite bit of the symbol in location $n_{i+1}$.

Since both $s_{\text{sub}}$ and $s$ consist of just 1-runs and 2-runs, this puts some restrictions on the embedding of $s_{\text{sub}}$ in $s$, e.g., we cannot have $n_{i+1} - n_i > 3$ (all locations between them (and maybe longer) are identical and hence give a run that is too long). In particular, and more formally, we have the following restrictions

1. The first bit in $s_{\text{sub}}$ must be located as the first bit in $s$. This is because the first bit in $s$ must be a 1 bit. I.e., $n_1 = 1$.

2. Let $b_i$ be a 1-run in $s_{\text{sub}}$ and assume w.l.o.g. that it is a 0 bit. Its location, $n_i$, must be chosen such that the location of the next bit in $s_{\text{sub}}$, $b_{i+1}$, is either

   (a) $n_{i+1} = n_i + 1$. I.e., $\langle b_i, b_{i+1} \rangle = \langle 01 \rangle$ in $s_{\text{sub}}$ is mapped to $\langle 01 \rangle$ in $s$, or

   (b) $n_{i+1} = n_i + 2$. I.e., $\langle 01 \rangle$ in $s_{\text{sub}}$ is mapped to $\langle 001 \rangle$ in $s$.

   The case where $b_i = 1$ is completely analogous.

3. Let $b_i, b_{i+1}$ be a 2-run in $s_{\text{sub}}$ (i.e., the symbols of $b_i$ and $b_{i+1}$ are the same) and assume w.l.o.g. that both symbols are 0. The locations $n_i, n_{i+1}, n_{i+2}$ of $b_i, b_{i+1}, b_{i+2}$ ($b_{i+2}$ is a 1 bit) in $s$ must be chosen in accordance with one of the following cases:

   (a) $n_{i+1} = n_i + 1$ and $n_{i+2} = n_i + 2$. I.e., $\langle 001 \rangle$ in $s_{\text{sub}}$ is mapped to $\langle 001 \rangle$ is $s$.

   (b) $n_{i+1} = n_i + 2$ and $n_{i+2} = n_i + 3$. I.e., $\langle 001 \rangle$ in $s_{\text{sub}}$ is mapped to $\langle 0101 \rangle$ in $s$.

   (c) $n_{i+1} = n_i + 2$ and $n_{i+2} = n_i + 4$. I.e., $\langle 001 \rangle$ in $s_{\text{sub}}$ is mapped to $\langle 01001 \rangle$ in $s$.

   (d) $n_{i+1} = n_i + 3$ and $n_{i+2} = n_i + 4$. I.e., $\langle 001 \rangle$ in $s_{\text{sub}}$ maps to $\langle 01101 \rangle$ in $s$.

   (e) $n_{i+1} = n_i + 3$ and $n_{i+2} = n_i + 5$. I.e., $\langle 001 \rangle$ in $s_{\text{sub}}$ is mapped to $\langle 011001 \rangle$ in $s$.

   The case where $b_i = 1$ is completely analogous.

4. If $n_{m-\delta m} < m$, then the remaining bits of $s$ must be filled with 1-runs and 2-runs such that the total number of 1-runs and 2-runs is exactly $\beta_1 m$ and $\beta_2 m$, respectively.

It is not hard to verify that any arrangement that does not follow the restrictions above will either contain a run of length 3 or more, will not have the right number of 1-runs, or will not correspond to the lexicographically first embedding of $s_{\text{sub}}$ in $s$.

We shall think of the cases above as describing operations that can be performed on a string $s'$. E.g. if $s' = \langle 101001 \rangle$ and we apply 3e to the last three bits in $s'$ then we will get the string $\langle 1010\mathbf{11}0\mathbf{0}1 \rangle$, where the bold symbols are the symbols that were added from the application of 3e (in other words, the symbols that are not bold are the embedding of the original string). If we then apply, say, 2b to the second and third bits of the new string then we will get the string $\langle 10\mathbf{0}10\mathbf{11}0\mathbf{0}1 \rangle$ etc.

To simplify matters note that if we consider a 2-run in $s'$, say $\langle 001 \rangle$ and we wish to apply 3c on it, i.e. map it to $\langle 01001 \rangle$ in $s$, then we can think about this as first applying 3b to $\langle 001 \rangle$, obtaining the string $\langle 0101 \rangle$ and then applying to the last two bits 2b, getting the string $\langle 01001 \rangle$. I.e. we can simulate 3c by first applying 3b and then applying 2b. Similarly, we can simulate each of the operations 3d and 3e using 3b and then applying 2b to the appropriate bits (for 3e we need to apply 3b and then 2b to two different locations).

Using the above terminology, we next describe an algorithm that given a string $s_{\text{sub}}$ generates $s \in S_{m,\beta_1}$ such that $s_{\text{sub}}$ is a subsequence of $s$. The algorithm will first select a subset of the 2-runs in $s_{\text{sub}}$ and apply 3b to them. Then it will add more 1-runs to the resulting string, locating them to the right of the last bit. Finally, it will apply 2b to several 1-runs.

There is a delicate point that we wish to stress before giving the algorithm. In this last step we restrict the 1-runs to which we can apply 2b. To illustrate why the restriction is needed, consider the following example: Consider the string $\langle 001 \rangle$ and apply 3b to it. This generates the string $\langle 0\mathbf{1}01 \rangle$, where, as before, the bold symbols represent the symbols that were added in the embedding. If we now apply 2b to the first two bits then we would get $\langle 0\mathbf{01}01 \rangle$. This however, is not the first lexicographical embedding of $\langle 001 \rangle$ in $\langle 00101 \rangle$ (which is $\langle 001\mathbf{01} \rangle$). Thus, if we wish to construct a lexicographically first embedding of $s_{\text{sub}}$ in the resulting string $s$ then in the last step, where we apply 2b to several runs, we should never apply 2b to the first bits resulting from the application of 3b in the first step.

In view of the above discussion we say that a 1-run is *frozen* if it is the first bit of a substring that resulted from applying 3b. In other words, a 1-run is not frozen if it is either an original 1-run of $s_{\text{sub}}$, a 1-run that was added in the second step, or if it is the 2nd or 3rd bits generated by applying 3b (i.e. if we had $\langle 001 \rangle \rightarrow \langle 0101 \rangle$ then the non-frozen 1-runs are the bold bits $\langle 0\mathbf{10}1 \rangle$, and the last bit may also be non-frozen).

Let $r_1$ and $r_2$ be the number of 1-runs and 2-runs in $s_{\text{sub}}$ and let $x$ be an integer such that $0 \leq x \leq \delta m$ and $r_1 + r_2 + 2x \leq \beta m$.

**Algorithm 1:** Embed

```
input : $s_{\text{sub}} \in S$ such that $|s_{\text{sub}}| = m - \delta m$,
        and $0 \leq x \leq \delta m$ where $r_1 + r_2 + 2x \leq \beta m$
output: A string $s \in S_{m,\beta_1}$ such that $s_{\text{sub}}$ is a subsequence of $s$
[1] Select $x$ 2-runs in $s_{\text{sub}}$ and apply 3b to them.
    /* total number of 1-runs is $r_1 + 3x$ and of 2-runs is $r_2 - x$          */
    /* total number of non-frozen 1-runs is $r_1 + 2x$                          */
[2] Add $\beta m - r_1 - r_2 - 2x$ many 1-runs to the right of the string
    /* total number of runs is $\beta m$ and number of 1-runs is $\beta m - r_2 + x$ */
    /* total number of non-frozen 1-runs is                                     */
    /* $\beta m - r_1 - r_2 - 2x + r_1 + 2x = \beta m - r_2$                     */
[3] Select $\delta m - (\beta m - r_1 - r_2 - x)$ non-frozen 1-runs and apply 2b to each of them
    /* length of resulting string is exactly $m$                                */
```

**Claim 3.6.** *Algorithm 1 returns a string in $S_{m,\beta_1}$.*

*Proof.* Step 1 turns each of the chosen $x$ 2-runs into three 1-runs, only two of which are non-frozen. Hence, the number of 2-runs is $r_2 - x$, the number of 1-runs is $r_1 + 3x$ and the number of non-frozen 1-runs is $r_1 + 2x$.

Step 2 completes the number of runs to $\beta m$ by introducing $\beta m - r_1 - r_2 - 2x$ new 1-runs[7]. The total number of 2-runs did not change, the total number of 1-runs is now

$$(r_1 + 3x) + (\beta m - r_1 - r_2 - 2x) = \beta m - r_2 + x$$

and the number of non-frozen 1-runs is

$$(r_1 + 2x) + (\beta m - r_1 - r_2 - 2x) = \beta m - r_2 \ .$$

Step 3 turns $\delta m - (\beta m - r_1 - r_2 - x)$ 1-runs into 2-runs. We now show that this gives a string in $S_{m,\beta_1}$. For this we need to show that it has only 1-runs and 2-runs and the correct number of runs of each type. The fact that we only get 1- and 2-runs follows from the definition of our operations. Now, the resulting number of 1-runs is

$$\begin{aligned}(\beta m - r_2 + x) - (\delta m - (\beta m - r_1 - r_2 - x)) &= 2\beta m - \delta m - 2r_2 - r_1 \\ &= 2\beta m - \delta m - (m - \delta m) \\ &= 2\beta m - m \\ &= \beta_1 m \ ,\end{aligned}$$

where we have used the facts that $m - \delta m = |s_{\text{sub}}| = 2r_2 + r_1$, that $\beta = \beta_1 + \beta_2$ and that $m = \beta_1 m + 2\beta_2 m$. Similarly, the number of 2-runs is

$$\begin{aligned}(r_2 - x) + (\delta m - (\beta m - r_1 - r_2 - x)) &= r_1 + 2r_2 + \delta m - \beta m \\ &= m - \delta m + \delta m - (\beta_1 + \beta_2)m \\ &= \beta_2 m \ .\end{aligned}$$

[7]Note that since $r_1 + r_2 + 2x \leq \beta m$, this operation is well defined.

Note that by our construction, the string begins with a 1 and it also ends with a 1 as the total number of runs is odd (recall Remark 3.2). Thus, the resulting string is in $S_{m,\beta_1}$ as claimed. $\square$

The next claim shows that any $s \in S_{m,\beta_1}$ that contains $s_{\text{sub}}$ as a subsequence can be obtained from the algorithm for an appropriate choice of $0 \leq x \leq \delta m$.

**Claim 3.7.** *For any $s \in S_{m,\beta_1}$ that contains $s_{\text{sub}}$ as a subsequence, there exists an $0 \leq x \leq \delta m$ and appropriate choices for the different steps of the algorithm so that the resulting string is $s$.*

*Proof.* As the proof contains many tedious details, we leave some of the arguments to the reader. Let $s \in S_{m,\beta_1}$ and denote by $1 = n_1 < n_2 < \ldots < n_{m-\delta m} \leq [n]$ the embedding of $s_{\text{sub}}$ in $s$ that corresponds to the first lexicographic appearance of $s_{\text{sub}}$ in $s$. By the discussion above, there are restrictions on the values $n_i$. Specifically, when we embed a 1-run from $s_{\text{sub}}$ in $s$, then $n_i$ and $n_{i+1}$ must satisfy one of the rules 2a or 2b and when we embed a 2-run, we must follow one of the rules 3a–3e.

Observe that the locations $n_1, \ldots, n_{m-\delta m}$ determine how to embed $s_{\text{sub}}$ into $s$, simply by going over all the runs in $s_{\text{sub}}$ from left to right. This can be easily proved by induction. For example, assume that the first run of $s_{\text{sub}}$ is $\langle 110 \rangle$ and $s$ starts with $\langle 10010 \rangle$. Then $n_1 = 1$, $n_2 = 4$, and $n_3 = 5$ and this implies that we need to perform 3d to the first run in $s_{\text{sub}}$. Now, our algorithm, when going over the first 2-run in Step 1, returns the string $\langle 1010 \rangle$. Then, in Step 3 the first 0 is a non-frozen 1-run and therefore the algorithm performs 2b to this 0. This combination of 3b and 2b is equivalent to 3d, as required. Thus, the algorithm correctly embeds the first run of $s_{\text{sub}}$ into $s$. We now move to the next run etc.

Notice that by the description above, after embedding all the runs, it may be the case that $n_{m-\delta m} < m$. However, looking back at the algorithm, in Step 2 we add several 1-runs to the embedding. Then, in Step 3, it may be the case that some of these 1-runs will be turned to 2-runs by the algorithm (all the added 1-runs to the right of the string are non-frozen). This ensures that the number of runs in the resulting string after Step 3 is exactly $\beta m$ and that we still have a valid embedding of $s_{\text{sub}}$.

Finally, note that there is a value of $x$ that will make the algorithm embed $s_{\text{sub}}$ successfully. The proof in the previous paragraph determines $x$ exactly. It is the number of times that we applied 3b in the embedding. Clearly, this number is at most $\delta m$ as otherwise the size of the resulting string will be larger than $m$. Also note that $r_1+r_2+2x \leq \beta m$ since otherwise we create a string that contains more than $\beta m$ runs. Thus, $x$ as we just defined satisfies both requirements $0 \leq x \leq \delta m$ and $r_1 + r_2 + 2x \leq \beta m$. $\square$

To conclude, if we consider all possible values $x$ can take, and all the possibilities to perform the choices in the algorithm we get an upper bound on the number of strings $s \in S_{m,\beta_1}$ which contain $s_{\text{sub}}$ as a subsequence. We are now ready to prove Proposition 3.5.

*Proof of Proposition 3.5.* By the argument above it is enough to count the number of possibilities for $x$ and the number of possible choices made by the algorithm. For any choice of $x$, there are exactly $\binom{r_2}{x}$ ways of selecting $x$ many 2-runs in Step 1 of Algorithm 1. In Step 2 of the algorithm we have no freedom since we add the new 1-runs at the end of the string. Finally, in Step 3 we have $\binom{\beta m - r_2}{\delta m - (\beta m - r_1 - r_2 - x)}$ many ways to choose $\delta m - (\beta m - r_1 - r_2 - x)$ many 1-runs among the non-frozen 1-runs.

We first note that

$$\binom{\beta m - r_2}{\delta m - (\beta m - r_1 - r_2 - x)} \leq \binom{\beta m - r_2}{\delta m - x} .$$

Indeed, as $r_2 \leq \beta_2 m + \delta m$ it follows that $\beta m - r_2 \geq \beta_1 m + \delta m \geq 2\delta m$. In addition, observe that $\delta m - (\beta m - r_1 - r_2 - x) \leq \delta m - x$ as otherwise Algorithm 1 performs more than $\delta m$ operations, in contradiction. As the binomial coefficients are monotonically increasing up to $(\beta m - r_2)/2 \geq \delta m$ the inequality follows.

Hence, the total number of strings that can be obtained from the algorithm is upper bounded by

$$\sum_{x=0}^{\delta m} \binom{r_2}{x} \binom{\beta m - r_2}{\delta m - (\beta m - r_1 - r_2 - x)} \leq \sum_{x=0}^{\delta m} \binom{r_2}{x} \binom{\beta m - r_2}{\delta m - x} = \binom{\beta m}{\delta m} ,$$

where the equality follows by Vandermonde's identity (E.g., [Tuc94, Chapter 5.5, Identity (10)]). $\square$

Armed with Propositions 3.4 and 3.5 we now show the existence of an appropriate inner code.

**Proposition 3.8.** *Let $0 \leq \beta_1, \delta \leq 1$ be parameters. Let $\beta = \frac{1+\beta_1}{2}$. For every $\varepsilon > 0$ there is $M_\varepsilon$ so that for every $m > M_\varepsilon$ there is a set $C \subseteq S_{m,\beta_1}$ of size $|C| = 2^{\lfloor m\mathcal{R}_{\text{in}} \rfloor}$ where*

$$\mathcal{R}_{\text{in}} = \beta h\left(\frac{\beta_1}{\beta}\right) - (\delta + \beta) h\left(\frac{\delta}{\delta + \beta}\right) - \beta h\left(\frac{\delta}{\beta}\right) - \varepsilon ,$$

*such that for every $c \neq c' \in C$ it holds that any string $s_{\text{sub}} \in S$ that is a subsequence of both $c$ and $c'$ is of length $|s_{\text{sub}}| < m - \delta m$.*

*Proof.* We first note that the number of binary strings in $S_{m,\beta_1}$ is exactly $\binom{\beta m}{\beta_1 m}$ as we have $\binom{\beta m}{\beta_1 m}$ ways to arrange the $\beta_1 m$ 1-runs and the $\beta_2 m$ 2-runs.

The construction of $C$ is done greedily. We go over all strings $s \in S_{m,\beta_1}$ and add them to $C$ one by one as long as they do not share a common subsequence that is too long (from $S$) with any string that is already in $C$. Propositions 3.4 and 3.5 imply that any $s \in S_{m,\beta_1}$ contains at most $\binom{\beta m + \delta m}{\delta m}$ many subsequences of length $m - \delta m$ from $S$, and each such string is a subsequence of at most $\binom{\beta m}{\delta m}$ strings in $S_{m,\beta_1}$. Thus, whenever we add a string to $C$ we exclude at most

$$\binom{\beta m + \delta m}{\delta m} \cdot \binom{\beta m}{\delta m}$$

other strings from being in $C$. Therefore, our codebook contains at least

$$|C| \geq \frac{\binom{\beta m}{\beta_1 m}}{\binom{\beta m + \delta m}{\delta m}\binom{\beta m}{\delta m}} \geq 2^{m\left(\beta h\left(\frac{\beta_1}{\beta}\right) - (\delta+\beta) h\left(\frac{\delta}{\delta+\beta}\right) - \beta h\left(\frac{\delta}{\beta}\right)\right) - O(\log m)}$$

codewords, where the inequality follows by Lemma 2.13. Thus, for every $\varepsilon > 0$ there exists large enough $m > 0$ such that the constructed set $C \subset S_{m,\beta_1}$ is of size $2^{\lfloor m\mathcal{R}_{\text{in}} \rfloor}$ where $\mathcal{R}_{\text{in}} = \beta h\left(\frac{\beta_1}{\beta}\right) - (\delta+\beta)h\left(\frac{\delta}{\delta+\beta}\right) - \beta h\left(\frac{\delta}{\beta}\right) - \varepsilon$. □

By construction, the code $C$ can handle an adversary that, given a codeword $c \in C$, returns a subsequence $s_{\text{sub}} \in S$ of $c$ where $|s_{\text{sub}}| \geq m - \delta m$. That is, we can uniquely identify the original codeword $c$ from $s_{\text{sub}}$. Our next goal is to show that our code can handle the usual edit distance adversary. In other words, it can handle an adversary that performs any $\delta m$ insertion and deletion (and hence it is not bound to return a string in $S$). The key observation is that if we look at two different codewords $c, c' \in C$ and denote by $s$ a longest common subsequence between $c$ and $c'$ then it must be that there exists $s' \in S$ that is also a subsequence of $c$ and $c'$ and $|s| = |s'|$.

**Proposition 3.9.** *Let $C$ be the code constructed in Proposition 3.8. For any two codewords $c, c' \in C$ it holds that* $\text{ED}(c, c') > 2\delta m$.

*Proof.* Let $c \neq c' \in C$. Lemma 2.3 gives

$$\text{ED}(c, c') = |c| + |c'| - 2\left|\text{LCS}(c, c')\right| = 2m - 2\left|\text{LCS}(c, c')\right| \ .$$

Observe that if $s$ is a longest common subsequence of $c$ and $c'$ then there is a string $s_{\text{sub}} \in S$, such that $|s_{\text{sub}}| = |s|$ and $s_{\text{sub}}$ is also a common subsequence of $c$ and $c'$. Indeed, let $s$ be a longest common subsequence of $c$ and $c'$. Since both $c$ and $c'$ start and end with 1, $s$ also starts and ends with 1. Moreover, for every three consecutive, equal bits in $s$, we can flip the second bit and the resulting string will still be a common subsequence of maximal length (as neither $c$ nor $c'$ contain a run of length 3 or more). Repeating this we will get a string only containing 1-runs and 2-runs, i.e. a string in $S$.

As $C$ was constructed so that not two codewords in $C$ share a common subsequence (from $S$) of length larger or equal to $m - \delta m$, it follows that

$$\text{ED}(c, c') = 2m - 2\left|\text{LCS}(c, c')\right| > 2m - 2(m - \delta m) = 2\delta m \ .$$

□

**Remark 3.10.** *Note that $C$ can be constructed in time at most $O\left(2^{2m} \cdot m^2\right)$ as in the worst case we compute the edit distance between any two possible strings.*

# 4 Construction

In this section we give a construction of a code for the $\text{BDC}_p$. Throughout this section we fix $p$.

We repeat the high level description of the construction from Section 1.3 (and as depicted in Figure 1 on page 7). We first do code concatenation. As outer code we use the one given in [HS17, Theorem 1.1] (restated as Theorem 2.14 here). As the inner code we use the code constructed in Proposition 3.8. Then, in order to protect the concatenated codeword from a large number of deletions, we first place a *buffer* of zeros between every two consecutive inner codewords. Since the decoder first looks for the buffers in order to identify where an inner code starts and where it ends, this step helps to reduce the amount of synchronization errors in the outer code. Secondly, we *blow-up* the inner codewords by replacing every run of length 1 with a run of length $N_1$ and every run of length 2 with a run of length $N_2$, where the symbols of the runs are preserved. For example, $\langle 11 \rangle$ turns into $\langle 1^{N_2} \rangle$ and $\langle 0 \rangle$ is replaced with $\langle 0^{N_1} \rangle$. If we choose $N_1$ and $N_2$ appropriately, then (with high probability) the decoder will identify the original run length.

We now give a formal description of our construction.

**The parameters:** At this point, we do not specify the parameters explicitly. We prefer to first present the scheme and analyze it before optimizing the parameters. However, the order by which we choose the parameters is important as there are some dependencies among them. First, we choose $M_1, M_2, \beta_1, M_B, \delta_{\text{out}}$ to be fixed constants. One should have in mind that $M_1 < M_2$ are the quantities through which $N_1$ and $N_2$ are determined, i.e., they determine how we blow-up the different types of runs. Then we choose $\delta_{\text{in}}$ to be larger than some quantity $\gamma = \gamma(M_1, T, M_2, \beta_1)$ that we later define (see Proposition 5.1). At this point, we can compute the value of $\mathcal{R}_{\text{in}}$, the rate of the inner code, using Proposition 3.8. Then, we choose a small enough $\epsilon_{\text{out}}$ that determines the alphabet size of the outer code $C_{\text{out}}$. Denote with $\mathcal{R}_{\text{out}}$ the rate of $C_{\text{out}}$ and by $n$ its block length. Finally, we pick $m$, the block length of the inner code, to satisfy[8] $\Sigma = \{0,1\}^{m \cdot \mathcal{R}_{\text{in}}}$.

While this may seem a bit confusing the main thing to remember is that $\epsilon_{\text{out}}$ that was picked at the end, can be taken to be as small a constant as we wish, or, in other words, we can pick $m$ to be as large a constant as we wish. This is important as we will bound the probabilities of several bad events by expressions of the form $\exp(-\Omega(m))$ and it will be important for us to be able to pick $m$ large enough as to make all our estimates small.

**Encoding:** The process of encoding starts with the outer code. Given as input a message $x \in \Sigma^{\mathcal{R}_{\text{out}} n}$, we encode it with the code given in Theorem 2.14 to obtain an outer codeword $c^{(\text{out})} = (\sigma_1, \ldots, \sigma_n) \in C_{\text{out}} \subset \Sigma^n$. Then, every symbol in $c^{(\text{out})}$, $\sigma_i \in \Sigma = \{0,1\}^{m \cdot \mathcal{R}_{\text{in}}}$, is encoded using the inner code to a codeword that we denote $c^{(\text{in})}_{\sigma_i}$. We thus get a codeword in the concatenated code

$$\left(c^{(\text{in})}_{\sigma_1}, \ldots, c^{(\text{in})}_{\sigma_n}\right) \in C_{\text{out}} \circ C_{\text{in}} \ .$$

Now that we have a codeword in the concatenated code we add additional layers of encoding that are crucial for the decoding algorithm to succeed.

1. Every two adjacent inner codewords are separated by a buffer of zeros of length $N_B = \lceil M_B \cdot m/(1-p) \rceil$.

2. In every inner codeword, we replace every 1-run with a run of length $N_1 = \lceil M_1/(1-p) \rceil$ where the symbol of the run is preserved.

3. In every inner codeword, we replace every 2-run with a run of length $N_2 = \lceil M_2/(1-p) \rceil$ where the symbol of the run is preserved.

[8]When we choose parameters we make sure that $m \cdot \mathcal{R}_{\text{in}}$ is an integer.

After the buffering and blow-up process we have three different run lengths $\lceil M_B \cdot m/(1-p) \rceil, \lceil M_1/(1-p) \rceil$ and $\lceil M_2/(1-p) \rceil$. Note that the buffer's length is much larger than $\lceil M_1/(1-p) \rceil$ and $\lceil M_2/(1-p) \rceil$ since it grows with $m$.

**Block length and rate:** Note that as $C_{\text{in}}$ contains strings in $S_{m,\beta_1}$, each of the $n$ inner codewords becomes of length $\lceil M_1/(1-p) \rceil \cdot \beta_1 m + \lceil M_2/(1-p) \rceil \cdot \beta_2 m$. As we have $n-1$ buffers between codewords the total block length is

$$(\lceil M_1/(1-p) \rceil \cdot \beta_1 m + \lceil M_2/(1-p) \rceil \cdot \beta_2 m) \cdot n + \lceil M_B \cdot m/(1-p) \rceil \cdot (n-1) .$$

Since the input to the encoding is a string in $\Sigma^{\mathcal{R}_{\text{out}} n}$ (and recall that $|\Sigma| = 2^{\mathcal{R}_{\text{in}} m}$) the rate $\mathcal{R}$ of the construction is given by

$$\begin{aligned}
\mathcal{R} &= \frac{\log\left(|\Sigma|^{\mathcal{R}_{\text{out}} n}\right)}{\beta_1 \lceil M_1/(1-p) \rceil mn + \beta_2 \lceil M_2/(1-p) \rceil mn + \lceil M_B m/(1-p) \rceil (n-1)} \\
&\geq \frac{\mathcal{R}_{\text{in}} \mathcal{R}_{\text{out}}}{\beta_1 \lceil M_1/(1-p) \rceil + \beta_2 \lceil M_2/(1-p) \rceil + M_B/(1-p) + 1/m} \\
&\geq \frac{\mathcal{R}_{\text{in}} \mathcal{R}_{\text{out}}}{\beta_1 M_1/(1-p) + \beta_2 M_2/(1-p) + \beta + M_B/(1-p) + 1/m} \\
&= \frac{\mathcal{R}_{\text{in}} \mathcal{R}_{\text{out}} (1-p)}{\beta_1 M_1 + \beta_2 M_2 + \beta(1-p) + M_B + (1-p)/m} .
\end{aligned} \tag{1}$$

We can avoid the ceilings if we consider values of $p$ such that $\lceil M_1/(1-p) \rceil, \lceil M_2/(1-p) \rceil$ and $\lceil M_B m/(1-p) \rceil$ are integers. In this case, the rate is

$$\mathcal{R} \geq \frac{\mathcal{R}_{\text{in}} \mathcal{R}_{\text{out}} (1-p)}{\beta_1 M_1 + \beta_2 M_2 + M_B} . \tag{2}$$

**Run time analysis:** By Theorem 2.14, the outer code can be constructed in linear time. Constructing the inner code requires time at most $O\left(2^{2m} \cdot m^2\right)$ (see Remark 3.10). As $m = \log |\Sigma| / \mathcal{R}_{\text{in}}$, we get that constructing the inner code takes time $O(m^2 \cdot 2^{2m}) = |\Sigma|^{O(1)} = O_{\epsilon_{\text{out}}}(1)$, which is constant. Thus, as all encoding steps are done in linear time, the encoding time complexity is $O(n)$.

## 5 Correctness and Analysis

We first present the decoding algorithm and then prove its correctness. After that, we show how to choose the parameters to obtain Theorem 1.1.

Let $y$ be the binary string received after transmitting $\text{Enc}(x)$. The decoding procedure is given in Algorithm 2 in page 22. Observe that the algorithm depends on some integral parameter $T$. When analyzing the algorithm we will see what $T$ has to satisfy in order for the algorithm to decode successfully with high probability. For the time being it is enough to remember that $M_1 < T < M_2$.

Before proving the correctness of the algorithm we give its run time analysis.

**Algorithm 2:** Decode with threshold $T$

```
input  : Binary string $y$ which is the output of the $\text{BDC}_p$ on $\text{ENC}(x)$
output: A message $\tilde{x} \in \Sigma^k$
[1] /* Identifying buffers Step:                                          */
    Every run of zeros of length longer than $M_B \cdot m/2$ is identified as a buffer.
    /* Denote by $s_1, \ldots, s_t$ the strings between the identified buffers.
       */
[2] /* Threshold decoding step:                                           */
    for every $s_i$ do
        for every run in $s_i$ do
            if the length of the run is longer than $T$ then
                Decode it to a run of length 2
            else
                Decode it to a run of length 1
            end
        end
    end
    /* Let $\tilde{c}_1, \ldots, \tilde{c}_t$ be the strings obtained in this step.    */
[3] /* Inner code decoding step:                                          */
    Use brute-force decoding to decode each $\tilde{c}_i$ to get $\tilde{\sigma}_i$. Denote
     $\tilde{\sigma}^{\mathsf{out}} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n) \in \Sigma^n$
[4] /* Outer code decoding step:                                          */
    Run the decoding algorithm of the outer code on $(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_t)$ to obtain $\tilde{x}$
    Output $\tilde{x}$
```

Figure 2: Algorithm for decoding our code over $\text{BDC}_p$. The algorithm is assumed to know the parameters $k, n, m, M_B, T$ as well as $C_{\text{in}}$ and $C_{\text{out}}$.

**Run time analysis of Algorithm 2.** It is clear that Steps 1 and 2 take linear time. Step 3 runs the inner decoding algorithm $n$ times. As the inner decoding algorithm is a brute force operation that is run on strings of constant length it takes constant time. Thus, the first three steps of the decoding algorithm require linear time. In Step 4 we run the decoding algorithm of [HS17] (recall Theorem 2.14), that requires $O(n^2)$ time. Therefore, the entire decoding procedure is dominated by the last step which runs in time $O(n^2)$.

## 5.1 Correctness of Decoding Algorithm

In this section, we prove that Algorithm 2 succeeds with high probability.

**Proposition 5.1.** *Given $M_1, T, M_2, M_B, \beta_1, \delta_{\text{in}}, \epsilon_{\text{in}}, \delta_{\text{out}}$ (as described in Section 4) let*

*$Z_1 \sim \mathrm{Bin}(\lceil M_1/(1-p)\rceil, 1-p)$ and $Z_2 \sim \mathrm{Bin}(\lceil M_2/(1-p)\rceil, 1-p)$. Denote*

$$\begin{aligned}
P^{(1)\to(2)} &:= \Pr[Z_1 \geq T+1],\\
P^{(1)\to(0)} &:= \Pr[Z_1 = 0],\\
P^{(2)\to(1)} &:= \Pr[Z_2 \leq T],\\
P^{(2)\to(0)} &:= \Pr[Z_2 = 0]\ ,
\end{aligned}$$

*and define*

$$\gamma := \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1+\beta_2)\, P^{(1)\to(0)} + 4\beta_2 P^{(2)\to(0)} \tag{3}$$

*(the reason for this definition of $\gamma$ is revealed later). If $\gamma < \delta_{\mathrm{in}}$, then there exists $\epsilon_0 = \epsilon_0(M_1, T, M_2, M_B, \beta_1, \delta_{\mathrm{in}}, \delta_{\mathrm{out}})$ such that for every $\epsilon_{\mathrm{out}} < \epsilon_0$ the following holds. Let $x \in \Sigma^{\mathcal{R}_{\mathrm{out}} n}$ (where $|\Sigma| = O_{\epsilon_{\mathrm{out}}}(1)$) be a message and let $y$ be the string obtained after encoding $x$ using our code and transmitting it through the BDC$_p$. Then, Algorithm 2 returns $x$ with probability $1 - \exp(-\Omega(n))$ when given $y$ as input.*

Observe that as $\epsilon_0 = \epsilon_0(M_1, T, M_2, M_B, \beta_1, \delta_{\mathrm{in}}, \delta_{\mathrm{out}})$, it does not depend on $m$ and $n$. The rest of Section 5.1 is devoted to proving Proposition 5.1. We first discuss the structure of the proof and prove relevant lemmas. The actual proof is given at the end of this section.

Let $\sigma^{\mathsf{out}} = (\sigma_1, \ldots, \sigma_n) \in \Sigma^n$ be the result of encoding $x$ with the outer code. I.e. the first step before concatenating with our inner code. Let $c^{(\mathsf{in})}_{\sigma_i}$ be the result of encoding $\sigma_i$ with the inner code.

The decoding algorithm succeeds if the decoding procedure of the outer code, which is executed in Step 4 of the algorithm, outputs the correct message. This happens if $\mathrm{ED}(\sigma^{(\mathsf{out})}, \tilde{\sigma}^{(\mathsf{out})}) \leq \delta_{\mathrm{out}} n$. To prove that this holds with high probability, we classify the errors that can be introduced at each step of the algorithm and bound the probability that we get too many of them.

There are three error types that increase the edit distance between $\sigma^{(\mathsf{out})}$ and $\tilde{\sigma}^{(\mathsf{out})}$:

1. *Deleted buffer:* This happens when the channel deleted too many bits from a buffer so that less than $M_B m/2$ bits survived the channel, and we did not identify this buffer in Step 1 of the algorithm.

2. *Spurious buffer:* In this case the algorithm mistakenly identifies a buffer inside an inner codeword. This might happen if there are many consecutive runs of the symbol 1 that were deleted by the channel. As a result, a long run of the symbol 0 is created and the algorithm will mistakenly identify it as a buffer in Step 1.

3. *Wrong inner decoding:* Here the decoding of the inner code returns a different inner codeword. This error happens if the edit distance between an inner codeword $c^{(\mathsf{in})}_{\sigma_i}$ and the corresponding $\tilde{c}_j$ is larger[9] than $\delta_{\mathrm{in}} m$.

In the following subsections, we analyze each error type separately and show that each happens with probability $\exp(-\Omega(m))$ per inner codeword. Our analysis of the first two error types is similar to [GL18], but our analysis of the third case is different.

[9]Note that it may be the case that due to decoding errors, the $i$th inner codeword was interpreted as the $j$th codeword by the decoder (e.g. if a buffer was deleted or a spurious buffer was introduced).

### 5.1.1 Deleted Buffer

**Proposition 5.2.** *Let $r_B$ be a buffer in Enc$(x)$. The probability that the decoding algorithm fails to identify it as a buffer in Step 1 is at most* $\exp(-\Omega(m))$.

*Proof.* Recall that the length of a buffer is $\lceil M_B m/(1-p)\rceil$. Therefore the expected number of bits that survive the transmission through the BDC$_p$ is at least $M_B m$. The decoder misses a buffer if the number of buffer bits that survived the transmission is smaller than $M_B m/2$. Let $Z$ denote the random variable that corresponds to the number of bits that survived the transmission of $r_B$ through the BDC$_p$. Clearly, $Z \sim \mathrm{Bin}(\lceil M_B m/(1-p)\rceil, (1-p))$. By using the Chernoff bound given in Lemma 2.10, we get that this error happens with probability

$$\Pr\left[Z < \frac{M_B m}{2}\right] = \Pr\left[Z < \left(1-\frac{1}{2}\right) M_B m\right] < \exp\left(-\frac{1}{8} M_B m\right) .$$

□

### 5.1.2 Spurious Buffer

Recall that this can happen if many consecutive runs of the symbol 1 were deleted by the channel, so a long run of the symbol 0 is created. If the length of this long run is longer than $M_B m/2$ then the decoder mistakenly identifies it as a buffer.

**Proposition 5.3.** *Let $c^{(\mathsf{in})}_{\sigma_i}$ be an inner codeword. Denote by* $\mathrm{Blow}(c^{(\mathsf{in})}_{\sigma_i})$ *the string obtained by blowing up the runs in $c^{(\mathsf{in})}_{\sigma_i}$ according to the encoding procedure. The probability that the decoder in Step 1 identifies a buffer inside the string obtained by transmitting Blow$(c^{(\mathsf{in})}_{\sigma_i})$ through the BDC$_p$ is at most* $\exp(-\Omega(m))$.

*Proof.* We first compute the probability that a run is deleted. Recall that after encoding the message we transmit runs of length $\lceil M_1/(1-p)\rceil$ or $\lceil M_2/(1-p)\rceil$. The probability that all the bits from a run of length $\lceil M_1/(1-p)\rceil$ are deleted by the BDC$_p$ is

$$p^{\lceil M_1/(1-p)\rceil} \le p^{M_1/(1-p)} \le e^{-M_1}.$$

Equivalently, the probability that all the bits from a run of length $\lceil M_2/(1-p)\rceil$ are deleted by the BDC$_p$ is

$$p^{\lceil M_2/(1-p)\rceil} \le p^{M_2/(1-p)} \le e^{-M_2}.$$

Suppose that $\ell$ consecutive runs of the bit 1 are deleted. We consider two cases.

First, consider the case where $\ell > mM_B/4M_2$. The probability that exactly $\ell$ runs of the symbol 1 are deleted is at most (the highest probability is obtained when all the $\ell$ runs are $\lceil M_1/(1-p)\rceil$-runs)

$$p^{\lceil M_1/(1-p)\rceil \ell} \le \exp\left(-M_1 \ell\right) \le \exp\left(-M_1 M_B m/4M_2\right) = \exp\left(-\Omega\left(m\right)\right).$$

The probability that there exist $\ge mM_B/(4M_2)$ consecutive runs of the symbol 1 that are deleted in a word of length $m$ is at most $O(m^2)\cdot \exp\left(-\Omega\left(m\right)\right) = \exp(-\Omega(m))$ (we just need to pick the start and end point of the consecutive runs).

Now, if $\ell \leq mM_B/(4M_2)$ consecutive runs of 1's are deleted, then there are $\ell + 1$ runs of zeros that are merged to a single run. Suppose that all the merged runs were 2-runs (so that the length of the run of the symbol 0 that was created is maximized). Denote by $Z$ the random variable that corresponds to the number of bits that survived the transmission of these $\ell + 1$ runs. It holds that $Z \sim \mathrm{Bin}\left((\ell+1)\left\lceil M_2/\left(1-p\right)\right\rceil, 1-p\right)$ and

$$\begin{aligned}
\mathbb{E}\left[Z\right] &= (\ell+1)\left\lceil M_2/(1-p)\right\rceil (1-p) \\
&\leq (\ell+1)(M_2+1) \\
&\leq (mM_B/(4M_2)+1)(M_2+1) \\
&\overset{(*)}{\leq} \frac{M_B m + 4M_2}{3} \\
&< \frac{2}{5} M_B m
\end{aligned}$$

where inequality $(*)$ holds for $M_2 \geq 3$ and the last inequality holds for large enough[10] $m$. Thus, we get by the Chernoff bound that the probability that $Z \geq M_B m/2$ is

$$\Pr\left[Z \geq \frac{M_B m}{2}\right] = \Pr\left[Z \geq \left(1+\frac{1}{4}\right)\frac{2}{5}M_B m\right] \leq \exp\left(-\frac{1}{120}M_B m\right) .$$

Hence, the probability that specific $\ell \leq mM_B/(4M_2)$ consecutive runs of the symbol 1 were deleted and a spurious buffer was created is at most $\exp(-M_B m/120)$. Therefore, for such an $\ell$, the probability that there exists a spurious buffer in an inner codeword of length $m$ is at most $m^2 \cdot \exp(-M_B m/120) \leq \exp(-M_B m/240)$, for large enough $m$. $\square$

**Remark 5.4.** *Proposition 5.3 upper bounds the probability that a spurious buffer is identified in a inner codeword. However, it may be the case that the decoder identifies two or more spurious buffers inside a single inner codeword. This is not an issue as the maximal number of spurious buffer inside an inner codeword is* $\leq 2/M_B$, *and therefore the expected number of spurious buffers in an inner codeword is at most* $(2/M_B) \cdot \exp(-\Omega(m)) = \exp(-\Omega(m))$.

### 5.1.3 Wrong Inner Decoding

This is the most difficult case to analyze. The inner decoding procedure might output a wrong codeword when the edit distance between an inner codeword $c^{(\mathsf{in})}_{\sigma_i}$ and the corresponding word that was obtained at Step 2 of the algorithm, $\tilde{c}_j$, is larger than $\delta_{\mathrm{in}} m$. The next proposition shows that the probability of this event is exponentially small in $m$.

**Proposition 5.5.** *Assume the setting of Proposition 5.1. Let* $c^{(\mathsf{in})}_{\sigma_i}$ *be an inner codeword. Assume that the buffers before and after* $c^{(\mathsf{in})}_{\sigma_i}$ *were detected correctly and that there were no spurious buffers in between. Suppose that* $\tilde{c}_j$ *is the corresponding string obtained at Step 2 of the decoding algorithm on* $s_j$. *Then,*[11]

$$\Pr\left[\mathrm{ED}\left(c^{(\mathsf{in})}_{\sigma_i}, \tilde{c}_j\right) > \delta_{\mathrm{in}} m\right] \leq \exp(-\Omega(m)) .$$

[10] Recall that by the way that we choose our parameters we pick $m$ at the end so that we can make it as large a constant as we wish.

[11] Recall that we use a different index $j$ to indicate that it may be the case that spurious buffers were found earlier, in some other inner codeword, or that some earlier buffers were mistakenly deleted.

We prove this claim in the remainder of this subsection, but first we give some intuition and introduce some important notions. Recall that a run $r_j$ in an inner codeword is replaced with a run of length $N_1 = \lceil M_1/(1-p) \rceil$ or $N_2 = \lceil M_2/(1-p) \rceil$. Let $Z_j$ be the random variable corresponding to the number of bits from this blown-up run that survived the transmission through the $\mathrm{BDC}_p$. If $|r_j| = 1$ then, $Z_j \sim \mathrm{Bin}\left(\lceil M_1/(1-p) \rceil, 1-p\right)$. If $|r_j| = 2$ then, $Z_j \sim \mathrm{Bin}\left(\lceil M_2/(1-p) \rceil, 1-p\right)$. Intuitively, in Step 2 the algorithm reads every $Z_j$ and decides according to the threshold $T$ if $Z_j$ corresponds to a run of length 1 or 2. However, it may be the case that, say, $Z_{j+1} = 0$ and then the algorithm will mistakenly base its decision according to the value of $Z_j + Z_{j+2}$, etc. For example, consider an initial string $\langle 00100 \rangle$. After the blow-up, we transmit the string $\langle 0^{N_2} 1^{N_1} 0^{N_2} \rangle$. Suppose that the middle run (the run consisting of the symbol 1) was deleted by the channel. The decoder then faces a long run of 0's and treats it as a single run and in particular, it will decode it as $\langle 0 \rangle$ or $\langle 00 \rangle$, or even as a spurious buffer. This motivates the following definitions.

**Definition 5.6.** *When $Z_j = 0$ we say that $r_j$ was* deleted by the channel.

**Remark 5.7.** *We shall make a distinction between runs that were deleted by the channel and those that our algorithm "deleted" so whenever we refer to a deleted bit we will stress which process caused the deletion.*

**Definition 5.8.** *For every $j \in [\beta m]$, let $b_j$ be the bit appearing in $r_j$. We denote*

$$r'_j = \begin{cases} \langle b_j b_j \rangle & \text{if } Z_j > T \\ \langle b_j \rangle & \text{if } 0 < Z_j \le T \\ \langle \rangle & \text{if } Z_j = 0 \end{cases} .$$

In other words, $r'_j$ is what Step 2 of our decoding algorithm would output when given $Z_j$ as input. In particular, $\left|r'_j\right|$ can be $0, 1, 2$, depending on $Z_j$. Note that if $\left|r'_j\right| = 0$ then it means that the channel deleted the run.

For the next definition, we remind the reader that in our setting the total number of runs in an inner codeword (and hence also in a blown-up word) is $\beta_1 m + \beta_2 m = \beta m$.

**Definition 5.9.** *A set $I \subset [\beta m], |I| \ge 2$, is called a* maximal merged set *if the following conditions hold:*

1. *For every $i \in I$ it holds that $Z_i > 0$.*
2. *All the bits from $I$ are merged into one run.*
3. *There is no set $J$ such that $I \subsetneq J$ and the bits from $J$ are merged into one run.*

For example, consider the following consecutive runs that were sent through the channel $\langle 0^{N_2} 1^{N_1} 0^{N_1} 1^{N_2} 0^{N_1} 1^{N_1} 0^{N_2} \rangle$. Suppose that the third run and the fifth run were deleted by the channel and the rest of the runs were not deleted by the channel. The maximal merged set corresponding to this deletion pattern is $I = \{2, 4, 6\}$.

**Claim 5.10.** *Let $I \subset [\beta m]$ be a maximal merged set. Denote $j = \min I$ and $k = \max I$. Then, all the runs $r_{j+1}, r_{j+3}, \ldots, r_{k-1}$ were* deleted by the channel.

*Proof.* Assume w.l.o.g. that $r_j$ and $r_k$ are runs of the symbol 0. For every $i \in \{j+1, j+3, \ldots, k-1\}$, $r_i$ is a run of symbol 1 and must be deleted by the channel. Otherwise, $I$ will not be a merged set. $\square$

**Definition 5.11.** *Let $I \subset [\beta m]$ be a maximal merged set and set $j = \min I$. We denote by $\tilde{r}_j$ the result of Step 2 of our decoding algorithm on this merged run.*

**Remark 5.12.** *It is important to remember that $r_j$ is the original run, $r'_j$ is what the algorithm would return when given $Z_j$ as input, and $\tilde{r}_j$ is what the algorithm actually returns when reading the bits of the merged run.*

We can now see that some bits that survived the channel were deleted by our algorithm as it failed to realize that they came from different runs. This is captured by the next definition.

**Definition 5.13.** *Let $I \subset [\beta m]$ be a maximal merged set. We say that the* decoding algorithm deleted $\left|r'_j\right| - |\tilde{r}_j| + \sum_{i \in I \setminus \{j\}} |r_i|$ *bits in the set $I$.*

As $\left|r'_j\right| \leq |\tilde{r}_j|$ the following claim is obvious.

**Claim 5.14.** *Let $I$ be a maximal merged set and set $j = \min I$. The number of bits deleted by the decoding algorithm in the merged set $I$ is at most $\sum_{i \in I \setminus \{j\}} |r_i|$.*

We next extend Claim 5.14 and bound the total number of bits that our algorithm deletes in an inner codeword. We assume that the buffers before and after the word were correctly identified by the decoding algorithm in Step 1.

**Claim 5.15.** *Let $D \subset [\beta m]$ be the indices of the runs that were deleted by the channel. If the last run was not deleted, i.e., $\beta m \notin D$, then the number of bits that were deleted by the decoding algorithm is at most $\sum_{i \in D} |r_{i+1}|$.*

*If the last run was deleted by the channel, i.e., $\beta m \in D$, then the number of bits deleted by the algorithm is at most $\sum_{i \in D \setminus \{\beta m\}} |r_{i+1}| + 2$.*

*Proof.* We first deal with the case where some runs were merged with the bits in the buffers (before or after the word). This happens if the first or the last run were deleted by the channel. If $1 \in D$ then let $r_{i'}$ to be the first run of the symbol 1 that was not deleted by the channel. Then, all runs of the symbol 0 before $r_{i'}$ were merged to the left buffer. Therefore, $D_L := \{1, 3, \ldots, i'-2\} \subseteq D$ and the decoding algorithm deleted exactly $|r_2| + \ldots + |r_{i'-1}| = \sum_{\ell \in D_L} |r_{\ell+1}|$ bits (since all these runs were considered as part of the buffer).

Similarly, if $\beta m \in D$ define $r_{i'}$ to be the last run of the symbol 1 that was not deleted by the channel. In this case all the runs of 0's after $r_{i'}$ were merged to the right buffer. In this case, $D_R := \{i'+2, i'+4, \ldots, \beta m\} \subseteq D$ and the decoding algorithm deleted exactly $|r_{i'+1}| + \ldots + |r_{\beta m-1}| \leq 2 + \sum_{\ell \in D_R \setminus \{\beta m\}} |r_{\ell+1}|$ bits.

We now account for inner deletions (i.e., those that did not cause runs to merge with buffers). These deletions may generate what we called maximal merged sets. Let $I_1, \ldots, I_t$ be all maximal merged sets, excluding those that were merged with buffers. Denote $j_i = \min I_i$ and $k_i = \max I_i$ and let[12] $D_i := D \cap [j_i + 1, k_i - 1]$ for $i \in [t]$.

[12]Note that it may be the case that the set $D' := D \setminus (\cup_i D_i)$ is not the empty set. In this case the indices in $D'$ correspond to runs that were deleted by the channel but did not cause a merge. E.g., if two consecutive runs are deleted by the channel and the runs before and after were not deleted, then this does not make our algorithm to delete additional bits.

According to Claim 5.10 it holds that $\{j_i+1, j_i+3, \ldots, k_i-1\} \subseteq D_i$. Thus, $I_i \subseteq \{j_i, j_i+2, \ldots, k_i\}$. Claim 5.14, implies that the number of bits deleted by the algorithm in $I_i$ is at most $\sum_{\ell \in I_i \setminus \{j_i\}} |r_\ell|$. Thus, the total number of bits deleted by the algorithm, excluding those bits from $D_L \cup D_R$, is bounded from above by

$$\sum_{i=1}^{t} \sum_{\ell \in I_i \setminus \{j_i\}} |r_\ell| \le \sum_{i=1}^{t} \sum_{\ell \in D_i} |r_{\ell+1}| \le \sum_{i \in D \setminus (D_L \cup D_R)} |r_{i+1}| \,.$$

Taking into account the deleted bits from $D_L \cup D_R$ the claim follows. □

We now use concentration bounds to argue about the expected number of bits that were deleted and the effect on the edit distance between the original inner codeword and the one returned by the algorithm in Step 2.

We first study the probability that $r_j \neq r'_j$ (recall Definition 5.8):

1. If $|r_j| = 1$ then there are two possible types of errors:

   (a) $|r'_j| = 2$: We denote the probability for this to happen by

   $$P^{(1)\to(2)} := \Pr[Z_j \ge T+1] \,.$$

   We next give two estimates of this probability, one is an exact calculation and the other is an upper bound. Direct calculation gives

   $$P^{(1)\to(2)} = \Pr[Z_j \ge T+1] = \sum_{i=T+1}^{\left\lceil \frac{M_1}{1-p} \right\rceil} \binom{\left\lceil \frac{M_1}{1-p} \right\rceil}{i} (1-p)^i \cdot p^{\left\lceil \frac{M_1}{1-p} \right\rceil - i} \,. \tag{4}$$

   We next would like to use the Poisson distribution to give a simpler bound. For this we would like to use Theorems 2.8 and 2.9.

   **Lemma 5.16.** *Let $q \ge 1-p$ and $T \ge M_1 + q$. It holds that*

   $$P^{(1)\to(2)} \le 1 - e^{-M_1 - q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!} \tag{5}$$

   *Moreover, the function $f(q) := 1 - e^{-M_1-q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!}$ is monotonically increasing in $q$.*

   *Proof.* Define $Y(j,x) \sim \mathrm{Bin}(j, (M_1+x)/j)$. Observe that $\mathbb{E}[Y(j,x)] = M_1 + x$. Denote $n' = \lceil M_1/(1-p) \rceil$. First note that

   $$\Pr[Z_j \ge T+1] \le \Pr[Y(n', 1-p) \ge T+1]$$

   since the expectation of $Y(n', (1-p))$ is $M_1 + (1-p)$ whereas the expectation of $Z_j$ is $\le M_1 + (1-p)$ and they are both binomial distributions on $n'$ trials. By the same reasoning we have that for every $j \ge n'$

   $$\Pr[Y(j, (1-p)) \ge T+1] \le \Pr[Y(j,q) \ge T+1] \,.$$

Let $P(x) \sim \text{Poisson}(x)$. Theorem 2.8 implies that $\lim_{j\to\infty} Y(j,x) = P(M_1+x)$. Therefore,

$$\begin{aligned} P^{(1)\to(2)} = \Pr\left[Z_j \geq T+1\right] &\leq \Pr[Y(n',q) \geq T+1] \\ &= 1 - \Pr\left[Y(n',q) \leq T\right] \\ &\leq 1 - \lim_{j\to\infty} \Pr\left[Y(j,q) \leq T\right] \\ &= 1 - \Pr\left[P(M_1+q) \leq T\right] \\ &= 1 - e^{-M_1-q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!} \ , \end{aligned}$$

where the second inequality follows from Theorem 2.9 due to monotonicity for $T \geq M_1 + q$.

Note that the monotonicity of $f(q) = 1 - e^{-M_1-q} \sum_{i=0}^{T} \frac{(M_1+q)^i}{i!}$ follows from Lemma 2.7. □

(b) $\left|r'_j\right| = 0$: Here the blown-up run was completely deleted by the channel. The probability for this to happen is $P^{(1)\to(0)} := \Pr[Z_j = 0]$. It holds that,

$$P^{(1)\to(0)} = \Pr[Z_j = 0] = p^{\left\lceil \frac{M_1}{1-p} \right\rceil}. \tag{6}$$

It also holds that for any $p \in (0,1)$,

$$P^{(1)\to(0)} = \Pr[Z_j = 0] \leq e^{-M_1}. \tag{7}$$

2. Similarly, when $|r_j| = 2$ there are two cases to consider:

(a) $\left|r'_j\right| = 1$: The probability for this to happen is $P^{(2)\to(1)} := \Pr[Z_j \leq T]$. As before, the exact calculation is

$$P^{(2)\to(1)} = \Pr[Z_j \leq T] = \sum_{i=0}^{T} \binom{\left\lceil \frac{M_2}{1-p} \right\rceil}{i} (1-p)^i \cdot p^{\left\lceil \frac{M_2}{1-p} \right\rceil - i} \ . \tag{8}$$

Similarly to the calculations for $P^{(1)\to(2)}$, we would like to upper bound $P^{(2)\to(1)}$ using a simpler expression coming from the Poisson distribution.

**Lemma 5.17.** *For every $p$ and for $T \leq M_2 - 1$, it holds that*

$$P^{(2)\to(1)} \leq e^{-M_2} \sum_{i=0}^{T} \frac{M_2^i}{i!} \ . \tag{9}$$

*Moreover, for every $q \geq p$ such that $M_2/(1-q)$ is an integer, it holds that*

$$P^{(2)\to(1)} \leq \sum_{i=0}^{T} \binom{\frac{M_2}{1-q}}{i} (1-q)^i \cdot q^{\frac{M_2}{1-q} - i} \tag{10}$$

*Proof.* For a natural number $1 \leq i$, let $Y(i) \sim \text{Bin}(i, M_2/i)$. Let $P \sim \text{Poisson}(M_2)$. Observe that $\Pr[Z_j \leq T] \leq \Pr[Y(\lceil M_2/(1-p)\rceil) \leq T]$ as the latter can only have smaller expectation. Since $\lim_{i\to\infty} Y(i) \sim P$ and due to the monotonicity implied by Theorem 2.9 we get that when $T \leq M_2 - 1$, for every $p$ it holds that:

$$\begin{aligned}
P^{(2)\to(1)} = \Pr[Z_j \leq T] &\leq \Pr[Y(\lceil M_2/(1-p)\rceil) \leq T] \\
&\leq \Pr[Y(\lceil M_2/(1-q)\rceil) \leq T] \\
&= \Pr[Y(M_2/(1-q)) \leq T] \\
&\leq \lim_{i\to\infty} \Pr[Y(i) \leq T] \\
&= \Pr[P \leq T] \\
&= e^{-M_2} \sum_{i=0}^{T} \frac{M_2^i}{i!} \ ,
\end{aligned}$$

where the second and the third inequalities hold due to Theorem 2.9 for $T \leq M_2 - 1$. Note that the second inequality proves the second statement in the lemma. □

(b) $|r'_j| = 0$: The probability for this to happen is $P^{(2)\to(0)} := \Pr[Z_j = 0]$. It holds that,

$$P^{(2)\to(0)} = \Pr[Z_j = 0] = p^{\lceil \frac{M_2}{1-p} \rceil} \tag{11}$$

and for every $p \in (0,1)$ we have

$$P^{(2)\to(0)} = \Pr[Z_j = 0] \leq e^{-M_2} \ . \tag{12}$$

Recall that $c^{(\text{in})}_{\sigma_i}$ is an inner codeword that consists of exactly $\beta_1 m$ 1-runs and $\beta_2 m$ 2-runs. Also recall that we blow-up an inner codeword, $c^{(\text{in})}_{\sigma_i}$, and send it through the $\text{BDC}_p$. Suppose that Step 1 of the algorithm identified the $i-1$'th and the $i$'th buffer and that there were no spurious buffers in between. Let $s_j$ be the binary string corresponding to this decoding window obtained in Step 1, and let $\tilde{c}_j$ be the result of Step 2 of the algorithm on $s_j$.

For every $j \in [\beta m - 1]$, Let $X_j$ be the random variable defined by

$$X_j = \begin{cases} 0 & \text{if } |r_j| = |r'_j| \\ 1 & \text{if } Z_j > 0 \text{ and } |r_j| \neq |r'_j| \\ |r_j| + |r_{j+1}| & \text{if } Z_j = 0 \end{cases} .$$

Similarly define $X_{\beta m}$ to be

$$X_{\beta m} = \begin{cases} 0 & \text{if } |r_{\beta m}| = |r'_{\beta m}| \\ 1 & \text{if } Z_{\beta m} > 0 \text{ and } |r_{\beta m}| \neq |r'_{\beta m}| \\ |r_{\beta m}| + 2 & \text{if } Z_{\beta m} = 0 \end{cases} .$$

**Claim 5.18.** *Let $c^{(\text{in})}_{\sigma_i}$ be an inner codeword. Assume that the buffers before and after $c^{(\text{in})}_{\sigma_i}$ were detected correctly and assume that there were no spurious buffers in between.*

*Suppose that $\tilde{c}_j$ is the corresponding string obtained at Step 2 of the decoding algorithm on $s_j$. Then,*

$$\mathrm{ED}\left(c^{(\mathrm{in})}_{\sigma_i}, \tilde{c}_j\right) \le \sum_{j=1}^{\beta m} X_j \ .$$

*Proof.* If $r_j$ is a 1-run and $r'_j$ is a 2-run then there was an insertion. Equivalently, if $r_j$ is a 2-run and $r'_j$ is a 1-run then there was a deletion. If a run was completely deleted by the channel then according to Claim 5.15, at the worst case scenario, the following run is also deleted by the algorithm. The definition of the $X_j$'s accounts for all that. □

Note that we may do over counting in some scenarios, e.g., if $r_{j+1} \neq r'_{j+1}$ and $r_j$ was deleted by the channel then $X_j + X_{j+1} = |r_j| + |r_{j+1}| + 1$ but the edit distance is at most $|r_j| + |r_{j+1}|$. This over counting makes the upper bound less tight.

Set $X = \sum_{j=1}^{\beta m} X_j$. We next upper bound and lower bound $\mathbb{E}[X]$.

**Claim 5.19.** *It holds that*

$$\mathbb{E}\left[X\right] \ge \xi m \ ,$$

*where*

$$\xi = \beta_1 \left(P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)}\right) + \beta_2 \left(P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)}\right) \ .$$

*Proof.* For every $X_j$ such that $r_j$ is a 1-run we have

$$\mathbb{E}[X_j] \ge 1 \cdot P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)} \ ,$$

where we used the fact that $|r_j| + |r_{j+1}| \ge 2$. Similarly, for every $X_j$ such that $r_j$ is a 2-run we have

$$\mathbb{E}[X_j] \ge 1 \cdot P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)} \ .$$

As there are exactly $\beta_1$ 1-runs and $\beta_2$ 2-runs, the claim follows. □

**Claim 5.20.** *It holds that*

$$\mathbb{E}\left[X\right] \le \gamma m + P^{(1)\to(0)} \ ,$$

*where,*

$$\gamma = \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1 + \beta_2) \cdot P^{(1)\to(0)} + 4\beta_2 \cdot P^{(2)\to(0)} \ , \tag{13}$$

*is the same $\gamma$ as in Proposition 5.1.*

For the proof we shall denote with $X_j^{i,k}$ the random variable $X_j$ when $r_j$ is an $i$-run and $r_{j+1}$ is a $k$-run.

*Proof.* Suppose that $r_{\beta m}$ is a 1-run. As will be explained later, this is the worst case, i.e., the upper bound that we prove on $\mathbb{E}[X]$ is largest in this case. Denote by $Y^{i,k} \subseteq [\beta m - 1]$ the set of indices $j \in [\beta m - 1]$ such that $r_j$ is an $i$-run and $r_{j+1}$ is a $k$-run. From linearity of expectation it follows that

$$\mathbb{E}[X] = \sum_{j\in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j\in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \sum_{j\in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j\in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] + \mathbb{E}\left[X_{\beta m}\right] \ .$$

Let $\lambda_1$ be such that $|Y^{1,2}| = \lambda_1 m$. Thus, $|Y^{1,1}| = (\beta_1 - \lambda_1)m - 1$ (where the 1 is subtracted because of the last run, which we assumed is a 1-run). By definition we have that

$$\sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] = \left(1 \cdot P^{(1)\to(2)} + 3 \cdot P^{(1)\to(0)}\right) \cdot \lambda_1 m$$

and

$$\sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] = \left(1 \cdot P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)}\right) \cdot ((\beta_1 - \lambda_1)m - 1) \ .$$

Observe that

$$\begin{aligned} & \sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \mathbb{E}[X_{\beta m}] \\ = & \left(1 \cdot P^{(1)\to(2)} + 3 \cdot P^{(1)\to(0)}\right) \cdot \lambda_1 m + \left(1 \cdot P^{(1)\to(2)} + 2 \cdot P^{(1)\to(0)}\right) \cdot ((\beta_1 - \lambda_1)m - 1) \\ & + \left(1 \cdot P^{(1)\to(2)} + 3 \cdot P^{(1)\to(0)}\right) \\ = & P^{(1)\to(2)} \cdot \beta_1 m + P^{(1)\to(0)} \cdot (2\beta_1 m + \lambda_1 m + 1) \ . \end{aligned}$$

As there are exactly $\beta_2 m$ 2-runs, it holds that $0 \le \lambda_1 \le \beta_2$. Hence, this sum is maximized for $\lambda_1 = \beta_2$. We thus have that

$$\begin{aligned} & \sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \mathbb{E}[X_{\beta m}] \\ \le & \beta_1 m P^{(1)\to(2)} + (2\beta_1 + \beta_2) m P^{(1)\to(0)} + P^{(1)\to(0)} \ . \end{aligned} \tag{14}$$

Similarly, let $\lambda_2$ be such that $|Y^{2,1}| = \lambda_2 m$. Thus, $|Y^{2,2}| = (\beta_2 - \lambda_2)m$. It holds that

$$\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] = \left(1 \cdot P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)}\right) \cdot \lambda_2 m$$

and

$$\sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] = \left(1 \cdot P^{(2)\to(1)} + 4 \cdot P^{(2)\to(0)}\right) \cdot (\beta_2 - \lambda_2) m \ .$$

Since the sum $\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right]$ is maximized for $\lambda_2 = 0$ we get,

$$\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] \le \beta_2 m P^{(2)\to(1)} + 4\beta_2 m P^{(2)\to(0)} \ . \tag{15}$$

Combining (14) and (15) we obtain

$$\begin{aligned} \mathbb{E}[X] &= \sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] + \mathbb{E}[X_{\beta m}] + \sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] \\ &\le \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1 + \beta_2) \cdot P^{(1)\to(0)} + 4\beta_2 \cdot P^{(2)\to(0)} + P^{(1)\to(0)} \\ &= \gamma m + P^{(1)\to(0)} \ , \end{aligned} \tag{16}$$

as claimed.

Note that if $r_{\beta m}$ was a 2-run, then $|Y^{1,2}| + |Y^{1,1}| = \beta_1 m$ (no need to subtract 1 since the last run is now a 2-run) and we have,

$$\sum_{j \in Y^{1,2}} \mathbb{E}\left[X_j^{1,2}\right] + \sum_{j \in Y^{1,1}} \mathbb{E}\left[X_j^{1,1}\right] \leq \beta_1 m P^{(1)\to(2)} + (2\beta_1 + \beta_2) m P^{(1)\to(0)} \ .$$

In this case, we have that $|Y^{2,1}| + |Y^{2,2}| = \beta_2 m - 1$. Thus, if we let $\lambda_2$ be such that $|Y^{2,1}| = \lambda_2 m$ and $|Y^{2,2}| = (\beta_2 - \lambda_2)m - 1$ then

$$\begin{aligned} & \sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] + \mathbb{E}[X_{\beta m}] \\ = \ & \left(1 \cdot P^{(2)\to(1)} + 3 \cdot P^{(2)\to(0)}\right) \cdot \lambda_2 m + \left(1 \cdot P^{(2)\to(1)} + 4 \cdot P^{(2)\to(0)}\right) \cdot ((\beta_2 - \lambda_2)m - 1) \\ & + \left(1 \cdot P^{(2)\to(1)} + 4 \cdot P^{(2)\to(0)}\right) \\ = \ & P^{(2)\to(1)} \cdot \beta_2 m + P^{(2)\to(0)} \cdot (4\beta_2 m - \lambda_2 m) \ , \end{aligned}$$

and this sum is maximized for $\lambda_2 = 0$. We thus have that

$$\sum_{j \in Y^{2,1}} \mathbb{E}\left[X_j^{2,1}\right] + \sum_{j \in Y^{2,2}} \mathbb{E}\left[X_j^{2,2}\right] + \mathbb{E}[X_{\beta m}] \leq P^{(2)\to(1)} \beta_2 m + 4P^{(2)\to(0)} \beta_2 m$$

Then, if $r_{\beta m}$ is a 2-run we have

$$\begin{aligned} \mathbb{E}[X] &\leq \beta_1 m P^{(1)\to(2)} + (2\beta_1 + \beta_2) m P^{(1)\to(0)} + P^{(2)\to(1)} \beta_2 m + 4P^{(2)\to(0)} \beta_2 m \\ &= \gamma m < \gamma m + P^{(1)\to(0)} \ . \end{aligned}$$

□

Thus, for any constant $\gamma' > \gamma$ there exist a constant $M_{\gamma'}$ such that for all $m > M_{\gamma'}$ it holds that

$$\mathbb{E}[X] \leq \gamma m + P^{(1)\to(0)} < \gamma' m \ .$$

In the following claim we use concentration bound to show that the probability that $X$ is greater than $\gamma' m$, for $\gamma' > \gamma$, is exponentially small in $m$ and then we conclude that decoding of an inner codeword succeeds with high probability.

**Claim 5.21.** *For any $\gamma' > \gamma$ and for every constant $\nu > 0$ it holds that for a large enough $m$,*

$$\Pr[X > (1+\nu)\gamma' m] < \exp\left(-\frac{\nu^2 \xi^2 m}{8\beta}\right) = \exp(-\Omega(m)) \ ,$$

*where $\xi$ is as in Claim 5.19.*

*Proof.* First note that

$$\Pr[X > (1+\nu)\gamma' m] \leq \Pr[X > (1+\nu)\mathbb{E}[X]] \ ,$$

where by Claim 5.20 the inequality holds for large enough $m$. The delicate point is to notice that the $X_j$'s are independent. This is because each $X_j$ is determined solely according to the value of $Z_j$ (indeed, its value only depends on whether $Z_j = 0$, $Z_j \leq T$ or $Z_j > T$), and the random variables $Z_j$'s are independent by the definition of the binary deletion channel. For every $X_j$ it holds that $0 \leq X_j \leq 4$ and if we set $t = \nu\mathbb{E}[X]$ and apply Theorem 2.12 then we get that

$$\begin{aligned}\Pr\left[X > (1+\nu)\mathbb{E}[X]\right] &< \exp\left(-\frac{2\nu^2(\mathbb{E}[X])^2}{\beta m \cdot 4^2}\right)\\ &\leq \exp\left(-\frac{2\nu^2(\xi m)^2}{16\beta m}\right)\\ &= \exp\left(-\frac{\nu^2\xi^2 m}{8\beta}\right) ,\end{aligned}$$

where the second inequality follow from Claim 5.19.

□

We are now ready to prove the main claim of this subsection, Proposition 5.5.

*Proof of Proposition 5.5.* By Claim 5.18 $X$ is an upper bound on $\mathrm{ED}(c^{(\mathrm{in})}_{\sigma_i}, \tilde{c}_j)$. Thus,

$$\Pr\left[\mathrm{ED}\left(c^{(\mathrm{in})}_{\sigma_i}, \tilde{c}_j\right) > \delta_{\mathrm{in}} m\right] \leq \Pr[X > \delta_{\mathrm{in}} m] .$$

By the assumption in Proposition 5.1 we have that $\delta_{\mathrm{in}} > \gamma$. We thus get that

$$\Pr\left[X > \delta_{\mathrm{in}} m\right] = \Pr\left[X > \left(1 + \frac{\delta_{\mathrm{in}} - \gamma}{\delta_{\mathrm{in}} + \gamma}\right)\frac{\delta_{\mathrm{in}} + \gamma}{2} m\right] \leq \exp\left(-\left(\frac{\delta_{\mathrm{in}} - \gamma}{\delta_{\mathrm{in}} + \gamma}\right)^2 \frac{\xi^2}{8\beta} m\right) ,$$

where the last inequality follows from Claim 5.21 by plugging $\nu = \frac{\delta_{\mathrm{in}} - \gamma}{\delta_{\mathrm{in}} + \gamma}$ and $\gamma' = \frac{\delta_{\mathrm{in}} + \gamma}{2}$. This completes the proof of Proposition 5.5. □

**Remark 5.22.** *Observe that all the parameters involved in the upper bound in Proposition 5.5, namely, $\gamma, \xi, \beta$ are independent of $m$. That is, they only depend on $\delta_{\mathrm{in}}, M_1, M_2, T$ and $\beta_1$.*

We are now ready to prove Proposition 5.1.

*Proof of Proposition 5.1.* We would like to show that with high probability, the edit distance between the original outer codeword $\sigma^{(\mathrm{out})}$ and the string $\tilde{\sigma}^{(\mathrm{out})}$, obtained after Step 3 of the decoding algorithm, is smaller than $\delta_{\mathrm{out}} n$. To prove this we shall analyze the contribution of each of the error types (deleted buffer, spurious buffer and wrong inner decoding) on the edit distance.

A deleted buffer causes two inner codewords to merge and thus be decoded incorrectly by the inner code's decoding algorithm. When considering the effect of this on the edit distance between $\sigma^{(\mathrm{out})}$ and $\tilde{\sigma}^{(\mathrm{out})}$, this introduces two deletions and one insertion. Similarly, a single spurious buffer introduces one deletion and two insertions, since an inner codeword is split into two parts. Likewise, $\ell$ spurious buffers inside an inner codeword introduce 1 deletion and possibly $\ell + 1$ insertions. A wrong inner decoding causes just one deletion and one insertion. Therefore, every error type increases the edit distance between the original outer codeword $\sigma^{(\mathrm{out})}$ and $\tilde{\sigma}^{(\mathrm{out})}$ by at most three.

As mentioned, the outer decoding algorithm fails if $\mathrm{ED}\left(c^{(\mathsf{out})}, \tilde{c}^{(\mathsf{out})}\right) > \delta_{\mathrm{out}} n$. Thus, for this to happen, at least one of the following bad events must occur:

1. There were at least $\delta_{\text{out}} n/9$ deleted buffers.
2. There were at least $\delta_{\text{out}} n/9$ spurious buffers.
3. There were at least $\delta_{\text{out}} n/9$ inner codewords that were decoded incorrectly even though they did not have spurious buffers and their buffers were identified.

We first treat events (1) and (3). We saw in Propositions 5.2 and 5.5 that for every inner codeword, each error type happens with probability $\exp(-\Omega(m))$. Since $\delta_{\text{out}}$ is a fixed constant, there exists a large enough $m$ so that $\exp(-\Omega(m)) \leq \delta_{\text{out}}/10$ for each error type. An important observation is that, similarly to Remark 5.22, the constants in the $\exp(-\Omega(m))$ in the different propositions depend only on $\delta_{\text{in}}, \beta_1, M_1, T, M_2, M_B$ which are fixed constants and are not related to the outer code. Thus, we can choose a small enough $\epsilon_{\text{out}}$, which determines a large enough $m$, so that the probability for each error type is $\leq \delta_{\text{out}}/10$. By the Chernoff bound given in Lemma 2.10, for a large enough $n$, each of the two bad events happens with probability $\exp(-\Omega(n))$.

We now turn to event (2). By Proposition 5.3 and Remark 5.4, the number of spurious buffers in every inner codewrod is between 0 and $2/M_B$ and the expected number of spurious buffers in an inner codeword is $\exp(-\Omega(m))$. Again, we can choose a small enough $\epsilon_{\text{out}}$, which determines a large enough $m$, so that the expected number of spurious buffers in an inner codeword is $\leq \delta_{\text{out}}/10$. By the independence of the $\text{BDC}_p$, we can apply the Hoeffding bound (Theorem 2.12) and get that the probability that there are more than $\delta_{\text{out}} n/9$ spurious buffers is $\exp(-\Omega(n))$.

In conclusion, Algorithm 2 succeeds with probability $1 - \exp(-\Omega(n))$ as claimed. $\square$

## 5.2 Proof of Theorem 1.1

We now prove our main theorem.

*Proof of Theorem 1.1.* Our goal is to maximize the rate given in Equation (1) while assuring that the parameters that we pick guarantee successful decoding with high probability. Recall that the order by which we choose the parameters in our construction is the following. First, we choose $M_1, T, M_2, \beta_1, M_B, \delta_{\text{out}}$ to be fixed constants. Then, we compute upper bounds on $P^{(1)\to(2)}$, $P^{(1)\to(0)}$, $P^{(2)\to(1)}$, $P^{(2)\to(0)}$. Plugging these upper bounds to Equation (3), we get an upper bound[13] on $\gamma$ which we denote by $\tilde{\gamma}$. Note that $\tilde{\gamma}$ depends only on $M_1, T, M_2, \beta_1$, and $p$. Then we choose $\delta_{\text{in}}$ to be larger than $\tilde{\gamma}$, and in particular we have $\gamma \leq \tilde{\gamma} < \delta_{\text{in}}$. Proposition 5.1 guarantees that if we choose a small enough $\epsilon_{\text{out}}$, then our decoding algorithm will succeed with high probability. Thus, we only have to make sure that the rate that we get satisfies the statement in the theorem. We calculate the value of $\mathcal{R}_{\text{in}}$ using Proposition 3.8 and then use it to calculate the overall rate.[14]

We consider several regimes of $p$ and for each regime we choose suitable parameters.

**Case $p \geq 0.9$:** In this case we choose:

$$M_1 = 5.41, M_2 = 22.8, \beta_1 = 0.522, M_B = 10^{-5}, \delta_{\text{out}} = 2^{-20} \quad \text{and } \delta_{\text{in}} = 0.01052\ ,$$

[13]We do not compute the value of $\gamma$ exactly as it is too difficult to do parametrically.

[14]When applying Proposition 3.8, we set $\delta = \delta_{\text{in}}$.

and set $T = 12$. From Proposition 3.8 we get that, for our choice of parameters, the rate of the inner code is $\mathcal{R}_{\text{in}} = 0.5229$. The upper bounds for $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ are computed using Equations (7), (9), and (12). To upper bound $P^{(1)\to(2)}$, we use Equation (5) given in Lemma 5.16 with $q = 0.1$. Observe that as we assume $p \geq 0.9$ it follows that $q \geq 1 - p$.

One can plug in the upper bounds to Equation (3) and observe that $\tilde{\gamma} < \delta_{\text{in}}$. Proposition 5.1 guarantees that for a small enough $\epsilon_{\text{out}}$ our decoding algorithm succeeds with high probability. To calculate the rate we use Equation (1). For a large enough $m$ (e.g. $m > 10^5$) we obtain

$$\frac{0.5229(1-p)}{8.27323 + 0.761(1-p) + (1-p)/m} \geq \frac{0.5229(1-p)}{8.34933} > \frac{(1-p)}{16} .$$

**Case** $0.57 < p < 0.9$**:** For this regime we use the parameters

$$M_1 = 5.59, M_2 = 23.5, \beta_1 = 0.53, M_B = 10^{-5}, \delta_{\text{out}} = 2^{-20} \quad \text{and } \delta_{\text{in}} = 0.008013 ,$$

and set $T = 13$. We get that the rate of the inner code is $\mathcal{R}_{\text{in}} = 0.55224$. We first note that the calculations used to upper bound $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ were obtained by using Equations (6), (8) and (11) with $p = 0.9$. This can be done since Equations (6) and (11) are clearly monotonically increasing in $p$ and we are considering smaller values of $p$. Also, observe that since $M_2/(1-0.9) = 235$ is an integer, then by Equation (10) given in Lemma 5.17, for every $p \leq 0.9$,

$$P^{(2)\to(1)} \leq \sum_{i=0}^{T} \binom{\frac{M_2}{1-0.9}}{i} (1-0.9)^i \cdot (0.9)^{\frac{M_2}{1-0.9}-i} ,$$

which is exactly what we get from Equation (8) with $p = 0.9$. Now, to upper bound $P^{(1)\to(2)}$ we use Equation (5) with $q = 1 - 0.57$, which is fine as $p > 0.57$ and thus $q > 1 - p$. As before, calculations show that $\tilde{\gamma} < \delta_{\text{in}}$. Hence for a small enough $\epsilon_{\text{out}}$ our decoding algorithm succeeds with high probability by Proposition 5.1. Plugging the parameters into Equation (1) and letting $m$ be large enough we get

$$\frac{0.55224(1-p)}{8.48521 + 0.765(1-p) + (1-p)/m} > \frac{0.55224(1-p)}{8.81416} > \frac{1-p}{16} .$$

**Case** $0 < p \leq 0.57$**:** The parameters we choose for this regime are

$$M_1 = 5.59, M_2 = 20.21, \beta_1 = 0.53, M_B = 10^{-5}, \delta_{\text{out}} = 2^{-20} \quad \text{and } \delta_{\text{in}} = 0.006147 ,$$

and set $T = 13$. Using Proposition 3.8, we get that $\mathcal{R}_{\text{in}} = 0.577475$. As in the previous case, the upper bounds to $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ were obtained by using Equations (6), (8) and (11), this time with $p = 0.57$ (observe that $M_2/(1-0.57)$ is an integer). In this case $13 = T \geq \lceil M_1/(1-p) \rceil$. For a random variable $Z$ distributed as $Z \sim \text{Bin}\left(\lceil M_1/(1-p) \rceil, 1-p\right)$, it holds that

$$P^{(1)\to(2)} = \Pr[Z \geq T + 1] = 0$$

since bits can only be deleted by the $\mathrm{BDC}_p$.

One can simply verify that $\tilde{\gamma} < \delta_{\text{in}}$ and hence for a small enough $\epsilon_{\text{out}}$ our decoding algorithm succeeds with high probability by Proposition 5.1. Plugging the parameters into Equation (1) and letting $m$ be large enough we get that for the case $p \leq 0.57$, the rate of the construction is

$$\frac{0.57747(1-p)}{7.71206 + 0.765(1-p) + (1-p)/m} > \frac{0.57747(1-p)}{8.47706} > \frac{1-p}{16} \, .$$

This completes the proof of Theorem 1.1 □

## 6 Rates For Fixed Values of Deletion Probabilities

In Theorem 1.1 we constructed codes of rate larger than $(1-p)/16$ for the $\mathrm{BDC}_p$ that can be used for reliable communication. Note that even if $p \to 1$ our construction gives codes of positive rate. Now, we wish to fix $p$ (and thus leave the regime $p \to 1$) and instead of using the bounds given in Equations (5), (7), (9) and (12), we can use the exact direct calculations given in Equations (4), (6), (8) and (11), respectively. Using the exact bounds we can improve, for any fixed value of $p$, the rate of the code compared to what we obtained in Theorem 1.1. The reason that we can improve the bound is that in the proof of Theorem 1.1 we looked for a relatively simple argument that should work for every value of $p$. When $p$ is fixed, we can use more direct calculations to get a better bound. For example, we can get significant improvement by using Equation (6) instead of Equation (7). E.g., for $p = 0.8$ there is a relatively large difference between $p^{M_1/(1-p)}$ and $e^{-M_1}$. E.g., for $M_1 = 5$ we have that $e^{-5} = 0.00673$ and $0.8^{5/(0.2)} = 0.00377$. Such savings allow us to choose smaller value of $M_1$ for the case $p = 0.8$. Then, by reducing the value of $M_1$ we reduce also the values of $T$ and $M_2$ which eventually lead to an improved rate.

The reason that we do not optimize the calculation using these equations for every $p$ is that the optimization involves complex expressions involving all our parameters and it is not clear how to optimize it and get a closed formula for the rate for arbitrary $p$.

In [DM07], the authors gave constructions of probabilistic codes for the binary deletion channel. They derived lower bounds on the capacity of the $\mathrm{BDC}_p$ that are the best lower bounds as far as we know for fixed values of $p$.

In Table 1 we compare our results to the ones obtained in [DM07]. One can see that our rates are smaller by approximately a factor of 2. Yet, the construction presented in this paper is deterministic, has polynomial time complexity and has a simpler analysis.

Note that as $p$ tends to 1 the rate that we achieve approaches $(1-p)/15.7$ as can be seen in Figure 3.

**Remark 6.1.** *The calculations in Table 1 are done only for $p \geq 0.5$ as the focus of this paper is on large values of $p$. When $p$ is small we know that the rate is better than $(1-p)/9$ (it approaches $1-h(p)$) and one has to take a different approach in order to obtain good constructions.*

| $p$ | $(\beta_1, N_1, T, N_2, \mathcal{R}_{\text{in}}, \delta_{\text{in}})$ | Final rate | [DM07] |
|---|---|---|---|
| 0.50 | (0.497, 8, 7, 27, 0.5456, 0.00922) | 0.050682 | 0.10186 |
| 0.55 | (0.519, 9, 8, 34, 0.5525, 0.00825) | 0.043005 | 0.084323 |
| 0.60 | (0.508, 10, 8, 38, 0.5184, 0.01120) | 0.035935 | 0.069564 |
| 0.65 | (0.519, 13, 9, 49, 0.5545, 0.00810) | 0.029926 | 0.056858 |
| 0.70 | (0.509, 15, 9, 57, 0.5267, 0.01051) | 0.024353 | 0.045324 |
| 0.75 | (0.524, 20, 10, 75, 0.5400, 0.00910) | 0.019420 | 0.035984 |
| 0.80 | (0.514, 24, 10, 96, 0.5289, 0.01022) | 0.014830 | 0.027266 |
| 0.85 | (0.526, 34, 11, 138, 0.5413, 0.00895) | 0.010701 | 0.019380 |
| 0.90 | (0.537, 54, 12, 224, 0.5534, 0.00773) | 0.006845 | 0.012378 |
| 0.95 | (0.53, 108, 12, 452, 0.5402, 0.00893) | 0.003305 | 0.005741 |
| 0.99 | (0.52, 541, 12, 2280, 0.5318, 0.00985) | 0.000641 | - |

Table 1: Rates for fixed values of $p$. $N_1$ and $N_2$ are the lengths of the inner codeword runs after the blow-up. I.e., $N_1 = \lceil M_1/(1-p) \rceil$ and $N_2 = \lceil M_2/(1-p) \rceil$.

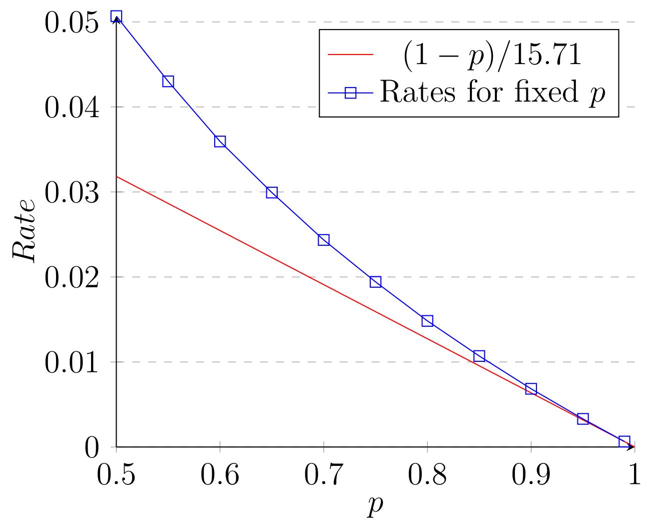


Figure 3: Rates for fixed values of $p$.

## 7 Poisson Repeat Channel

We first recall the definition of the PRC$_\lambda$.

**Definition 7.1.** *Let* $\lambda > 0$. *The* Poisson repeat channel with parameter $\lambda$ (PRC$_\lambda$) *replaces each transmitted bit randomly (and independently of other transmitted bits), with a discrete number of copies of that bit, distributed according to the Poisson distribution with parameter* $\lambda$.

This channel was first defined by Mitzenmacher and Drinea in [MD06] who used it to prove a lower bound of $(1-p)/9$ on the rate of the BDC. More recently, Cheraghchi [Che18] gave an upper bound on its capacity and showed further connections to the BDC.

Before proceeding, let us describe the connection between the PRC and the BDC discovered by Mitzenmacher and Drinea. What they observed is that a code for the PRC$_\lambda$ having rate $\mathcal{R}$, yields a code for the BDC$_p$ of rate $(1-p) \cdot \mathcal{R}/\lambda$. The reduction is via a probabilistic argument – from each codeword in the code for the PRC$_\lambda$ we generate a codeword for the BDC$_p$ as follows: we replace each of the bits in the codeword by a discrete number of copies of those bits, distributed according to the Poisson distribution with parameter $\lambda/(1-p)$. The intuition for the construction is that now, when we send the codeword through the BDC$_p$, the resulting word is distributed as if we had sent the original codeword through the PRC$_\lambda$.

To the best of our knowledge, prior to this work there were no explicit deterministic constructions of coding schemes for the PRC$_\lambda$. In this section, we prove that the scheme that we constructed for the BDC can also be used for PRC (with slightly different parameters). We note that one can also use the construction given in [GL18] to obtain a deterministic construction for the PRC, yet our construction yields better rates in this case as well.

We focus on the regime where $\lambda \leq 0.5$, as, in some sense, the PRC behaves like the BDC for small values of $\lambda$ – intuitively, the smaller $\lambda$ is the more likely deletions are.

We now describe the construction for this channel. Note that most of the details are identical to our construction for the BDC$_p$. Therefore, in order not to repeat the entire proof, we focus on the differences and leave the details to the reader.

## 7.1 Construction

We use the same inner and outer codes defined in Proposition 3.8 and Theorem 2.14. For parameters $M_1 < M_2$ and $M_B$ our construction is as follows:

**Encoding.** The only differences in the encoding procedure are the length of the buffers and the blow-up of the runs:

- We place a buffer of 0's between every two inner codewords, where the buffers length is $\lceil M_B m/\lambda \rceil$.
- Every run of length 1 is replaced with a run of length $\lceil M_1/\lambda \rceil$.
- Every run of length 2 is replaced with a run of length $\lceil M_2/\lambda \rceil$.

**Remark 7.2.** *We must choose $M_2 > \lambda$ since otherwise all runs in the inner code will be replaced with a run of length* 1.

**Decoding.** Since the inner and outer codes are the same we use the decoding algorithm given in Algorithm 2.

**Rate.** Similar to the calculations yielding Equation (1), the rate of this construction is

$$\begin{aligned} \mathcal{R} &= \frac{\log\left(|\Sigma|^{\mathcal{R}_{\text{out}} n}\right)}{\beta_1 \lceil M_1/\lambda \rceil nm + \beta_2 \lceil M_2/\lambda \rceil nm + \lceil M_B m/\lambda \rceil (n-1)} \\ &\geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}}}{\beta_1 \lceil M_1/\lambda \rceil + \beta_2 \lceil M_2/\lambda \rceil + M_B/\lambda + 1/m} \\ &\geq \frac{\mathcal{R}_{\text{out}}\mathcal{R}_{\text{in}} \cdot \lambda}{\beta_1 M_1 + \beta_2 M_2 + \beta\lambda + M_B + \lambda/m}\,. \end{aligned} \tag{17}$$

As before, we can avoid the ceilings if we consider values of $\lambda$ such that $\lceil M_1/\lambda \rceil$, $\lceil M_2/\lambda \rceil$ and $\lceil M_B m/\lambda \rceil$ are integers. In this case, the rate of the construction is given by

$$\mathcal{R} \geq \frac{\mathcal{R}_{\text{in}}\mathcal{R}_{\text{out}} \cdot \lambda}{\beta_1 M_1 + \beta_2 M_2 + M_B} \,. \tag{18}$$

## 7.2 Correctness of Decoding Algorithm

Since we use the same inner and outer codes in our encoding and the same decoding algorithm, the analysis performed in Section 5 can be repeated to this case as well with some minor modifications. We will briefly mention these modifications and leave the proofs to the reader.

We start by formally stating an analogous version of Proposition 5.1 to this setting.

**Proposition 7.3.** *Given $M_1, T, M_2, M_B, \beta_1, \delta_{\text{in}}, \epsilon_{\text{in}}, \delta_{\text{out}}$ (as described in Section 7.1) let $Z_1 \sim \text{Poisson}(\lambda \lceil M_1/\lambda \rceil)$ and $Z_2 \sim \text{Poisson}(\lambda \lceil M_2/\lambda \rceil)$. Denote*

$$\begin{aligned}
P^{(1)\to(2)} &:= \Pr[Z_1 \geq T+1] \,, \\
P^{(1)\to(0)} &:= \Pr[Z_1 = 0] \,, \\
P^{(2)\to(1)} &:= \Pr[Z_2 \leq T] \,, \\
P^{(2)\to(0)} &:= \Pr[Z_2 = 0] \,,
\end{aligned}$$

*and define*

$$\gamma := \beta_1 \cdot P^{(1)\to(2)} + \beta_2 \cdot P^{(2)\to(1)} + (2\beta_1 + \beta_2)\, P^{(1)\to(0)} + 4\beta_2 P^{(2)\to(0)} \,. \tag{19}$$

*Let $x \in \Sigma^{\mathcal{R}_{\text{out}} n}$ be a message and let $y$ be the string obtained after encoding $x$ using our code and transmitting it through the $PRC_\lambda$. If $\gamma < \delta_{\text{in}}$, then there exists $\epsilon_0 = \epsilon_0(M_1, T, M_2, M_B, \beta_1, \delta_{\text{in}}, \delta_{\text{out}})$ such that for every $\epsilon_{\text{out}} < \epsilon_0$ it holds that Algorithm 2 returns $x$ with probability $1 - \exp(-\Omega(n))$.*

Note that the only difference between this proposition and Proposition 5.1 is in the definitions of $Z_1$ and $Z_2$. Recall that the proof of Proposition 5.1 heavily relies on Propositions 5.2, 5.3, and 5.5. Therefore, to prove Proposition 7.3, one needs to formally state and prove analogous versions of Propositions 5.2, 5.3, and 5.5 in the setting of the PRC.

We first observe that it is very simple to prove the analogous claims to Propositions 5.2 and 5.3 by using Lemma 2.11 instead of Lemma 2.10 (since our random variables are now distributed according to the Poisson distribution). Hence we omit the details. We thus have that the probability of each error type is $\exp(-\Omega(m))$ per inner codeword. We focus on analyzing the case where we might output a wrong inner codeword in Step 3 of Algorithm 2 (i.e. the case analyzed in Proposition 5.5).

### 7.2.1 Wrong Inner Decoding

Note that as we consider the same threshold decoding step for decoding the inner windows (i.e., Step 2 in Algorithm 2) and the same inner code, the claims of Section 5.1.3 apply here as well. The difference from Section 5.1.3 is in the computations of the probabilities

$P^{(1)\to(2)}$ , $P^{(1)\to(0)}$, $P^{(2)\to(1)}$, $P^{(2)\to(0)}$. We focus on these computations as they play a significant role in computing the rate in Theorem 1.2.

Recall that in the encoding process, a run $r_j$ is replaced with a run of length $\lceil M_1/\lambda \rceil$ or $\lceil M_2/\lambda \rceil$ depending on $r_j$'s length. As in Section 5.1.3, define $Z_j$ to be the random variable corresponding to the number of bits from this blown-up run that survived the transmission through the PRC$_\lambda$. According to Lemma 2.6, $Z_j \sim \text{Poisson}(\lambda \lceil M_1/\lambda \rceil)$ if $|r_j| = 1$ and $Z_j \sim \text{Poisson}(\lambda \lceil M_2/\lambda \rceil)$ if $|r_j| = 2$. Let $r'_j$ be exactly as defined in Definition 5.8. As before, we study the probability that $r_j \neq r'_j$:

1. If $|r_j| = 1$ then there are two possible types of errors:

   (a) $|r'_j| = 2$: The probability for this to happen is $P^{(1)\to(2)} := \Pr[Z_j \geq T + 1]$. We next give two estimates, one is an exact calculation and the other is an upper bound. For every $\lambda$ we have

   $$\begin{aligned} P^{(1)\to(2)} &= \Pr[Z_j \geq T+1] \\ &= 1 - \Pr[Z_j \leq T] \\ &= 1 - e^{-\lambda \lceil \frac{M_1}{\lambda} \rceil} \sum_{i=0}^{T} \frac{(\lambda \lceil \frac{M_1}{\lambda} \rceil)^i}{i!} . \end{aligned} \tag{20}$$

   Let $Y$ be a random variable distributed as $Y \sim \text{Poisson}(M_1 + \lambda)$. We can upper bound $P^{(1)\to(2)}$ by

   $$\begin{aligned} P^{(1)\to(2)} = \Pr[Z_j \geq T+1] &= 1 - \Pr[Z_j \leq T] \\ &\leq 1 - \Pr[Y \leq T] \\ &= 1 - e^{-M_1-\lambda} \sum_{i=0}^{T} \frac{(M_1+\lambda)^i}{i!} . \end{aligned} \tag{21}$$

   where the inequality follows from Lemma 2.7 by noting that $\lambda \lceil M_1/\lambda \rceil \leq M_1 + \lambda$.

   (b) $|r'_j| = 0$: In this case, $r_j$ was completely deleted by the channel. The probability for this to happen is

   $$P^{(1)\to(0)} = \Pr[Z_j = 0] = e^{-\lambda \lceil M_1/\lambda \rceil} \leq e^{-M_1} . \tag{22}$$

2. If $|r_j| = 2$ then one of the following cases hold:

   - $|r'_j| = 1$: The probability for this to happen is $P^{(2)\to(1)} := \Pr[Z_j \leq T]$. As before, the exact probability calculation is

     $$P^{(2)\to(1)} = \Pr[Z_j \leq T] = e^{-\lambda \lceil \frac{M_2}{\lambda} \rceil} \sum_{i=0}^{T} \frac{(\lambda \lceil \frac{M_2}{\lambda} \rceil)^i}{i!} . \tag{23}$$

     Let $Y$ be a random variable distributed as $Y \sim \text{Poisson}(M_2)$ then it holds that

     $$P^{(2)\to(1)} = \Pr[Z_j \leq T] \leq \Pr[Y \leq T] = e^{-M_2} \sum_{i=0}^{T} \frac{M_2^i}{i!} , \tag{24}$$

     where the inequality follows from Lemma 2.7 by noting that $M_2 \leq \lambda \lceil M_2/\lambda \rceil$.

- $\left|r'_j\right| = 0$. The probability for this to happen is

$$P^{(2)\to(0)} = \Pr[Z_j = 0] = e^{-\lambda\lceil M_2/\lambda\rceil} \le e^{-M_2} \,. \tag{25}$$

By using these estimates and proceeding exactly as in the proof of Proposition 5.5 one gets that the probability of error in this case as well is $\exp(-\Omega(m))$. Combining everything together the proof of Proposition 7.3 follows similarly to the proof of Proportion 5.1 . In particular, Algorithm 2 decodes correctly in this setting as well.

#### 7.2.2 Proof of Theorem 1.2

As in the proof of Theorem 1.1, we first compute an upper bound on $\gamma$ (recall its definition in Proposition 7.3) that holds for all $\lambda \le 0.5$, then we compute the rate of the inner code by using Proposition 3.8 and finally we compute the rate of our code using Equation 17.

The parameters we use for our construction are

$$M_1 = 5.49, M_2 = 24.2, \beta_1 = 0.532, M_B = 10^{-5} \quad \text{and } \delta_{\text{out}} = 2^{-20} \,.$$

We pick $T = 13$ and set $\delta_{\text{in}} = 0.00954$.

First observe that for every $\lambda > 0$, we can upper bound $P^{(1)\to(0)}, P^{(2)\to(1)}, P^{(2)\to(0)}$ using Equations (22), (24), and (25) respectively. As we assume $\lambda \le 0.5$, we can upper bound $P^{(1)\to(2)}$ using Equation (21) with $\lambda = 0.5$ (due to monotonicity implied by Lemma 2.7). Plugging these upper bounds to Equation (19), we get an upper bound on $\gamma$ which, as before, we denote by $\tilde{\gamma}$. Calculating, it is simple to verify that $\gamma \le \tilde{\gamma} < \delta_{\text{in}}$. Therefore, for a small enough $\epsilon_{\text{out}}$, our decoding algorithm succeeds with high probability. Applying Proposition 3.8 we get an inner code of rate $\mathcal{R}_{\text{in}} = 0.53186$ and by letting $m$ be large enough, the rate of our concatenated code according to Equation (17) is

$$\mathcal{R} = \frac{0.5318\lambda}{8.58349 + 0.766\lambda + \lambda/m} > \frac{\lambda}{17} \,.$$

## 8 Open Questions

The main open question is to further improve the construction presented in this paper and close the gap to (and even surpass) the lower bound of $(1-p)/9$ on the capacity of the $\text{BDC}_p$. Alternatively, we can ask to come up with a deterministic construction for the $\text{PRC}_\lambda$ that gives better rates. By the reduction from the PRC to the BDC this will improve upon the constructions for the BDC.

Even though the capacity of the $\text{BDC}_p$ scales proportionally with $1-p$ for $p \to 1$, it is an interesting open question to understand if there is a constant $1/9 \le \mu \le 0.4143$ such that the capacity of the channel is $\mu(1-p)$ in the regime where $p$ is large.

Another interesting question is the maximal deletion fraction $\delta$, for which for every $\epsilon > 0$, there exists a code with rate bounded away from 0 that can handle $\delta - \epsilon$ fraction of adversarial deletions. One can easily see that $\delta = 1/2$ is an upper bound (we simply delete all 0's or all 1's). Recently, Guruswami, He, and Li [GHL21] showed that the upper bound is strictly less than one half, i.e., $\delta \le 1/2 - \delta_0$ where $\delta_0$ is extremely small. Bukh et al. [BGH17] showed that $\delta \ge \sqrt{2} - 1$. An interesting open question is to close the gap between the lower and upper bound.

## Acknowledgment

We wish to thank an anonymous reviewer for pointing out that our inner code is robust against the edit distance adversary and not just a restricted adversary considered in an earlier version of this paper, thus simplifying our arguments. We thank the anonymous reviewers for comments that improved the presentation of our results.

## References

[AS65] Theodore W Anderson and Stephen M Samuels. Some inequalities among binomial and poisson probabilities. In *Proc. Fifth Berkeley Symp. Math. Statist. Probab*, volume 1, pages 1–12, 1965.

[BGH17] Boris Bukh, Venkatesan Guruswami, and Johan Håstad. An improved bound on the fraction of correctable deletions. *IEEE Transactions on Information Theory*, 63(1):93–103, 2017.

[BLC$^+$16] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. *ACM SIGARCH Computer Architecture News*, 44(2):637–649, 2016.

[Che18] Mahdi Cheraghchi. Capacity upper bounds for deletion-type channels. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 493–506. ACM, 2018.

[CR03] Maxime Crochemore and Wojciech Rytter. *Jewels of stringology: text algorithms.* World Scientific, 2003.

[Dal11] Marco Dalai. A new bound on the capacity of the binary deletion channel with high deletion probabilities. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 499–502. IEEE, 2011.

[DM07] Eleni Drinea and Michael Mitzenmacher. Improved lower bounds for the capacity of iid deletion and duplication channels. *IEEE Transactions on Information Theory*, 53(8):2693–2714, 2007.

[Eli55] Peter Elias. Coding for noisy channels. *IRE Convention Record*, 4:37–46, 1955.

[GHL21] Venkatesan Guruswami, Xiaoyu He, and Ray Li. The zero-rate threshold for adversarial bit-deletions is less than 1/2. *arXiv preprint arXiv:2106.05250*, 2021.

[GL18] Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Transactions on Information Theory*, 2018.

[GRS12] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at http://www. cse. buffalo. edu/ atri/courses/coding-theory/book*, 2012.

[GW17] Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.

[Ham50] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.

[Hoe94] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[HS17] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the singleton bound. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 33–46. ACM, 2017.

[KMS10] Adam Kalai, Michael Mitzenmacher, and Madhu Sudan. Tight asymptotic bounds for the deletion channel with small deletion probabilities. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 997–1001. IEEE, 2010.

[Lev66] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[MBT10] Hugues Mercier, Vijay K Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys & Tutorials*, 12(1):87–96, 2010.

[MD06] Michael Mitzenmacher and Eleni Drinea. A simple lower bound for the capacity of the deletion channel. *IEEE Transactions on Information Theory*, 52(10):4657–4660, 2006.

[Mit09] Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge university press, 2005.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[Tuc94] Alan Tucker. *Applied combinatorics.* John Wiley & Sons, Inc., 1994.