Joint error correction enhancement of the Fountain Codes concept

Jarek Duda

Institute of Computer Science and Computer Mathematics, Jagiellonian University, Cracow, Poland,
email: dudajar@gmail.com

Abstract

Fountain Codes like LT or Raptor codes, also known as rateless erasure codes, allow to encode a message as some number of packets, such that any large enough subset of these packets is sufficient to fully reconstruct the message. Beside the packet loss scenario, the transmitted packets are usually damaged. Hence, an additional error correction scheme is often required: adding some level of redundancy to each packet to repair eventual damages. This approach requires a priori knowledge of the actual damage level: insufficient redundancy level denotes packet loss, overprotection means suboptimal channel rate. However, the sender may have inaccurate or even no a priori information about the final damage level, like for degradation of a storage medium or damage of picture watermarking.

This article introduces and discusses Joint Reconstruction Codes (JRC) to remove the need of a priori knowledge of damage level and sub-optimality caused by overprotection or discarding underprotected packets. It is obtained by combining both processes: reconstruction from multiple packets and forward error correction. Intuitively, instead of adding artificial redundancy to each packet, the packets are prepared to be simultaneously payload and redundancy. The decoder combines the resultant informational content of obtained packets accordingly to the actual noise level, estimated a posteriori independently for each packet. Assuming binary symmetric channel (BSC) of $\epsilon$ bit-flip probability, such potentially damaged bit carries $R_0(\epsilon) = 1 - h_1(\epsilon)$ bits of information, where $h_1$ is the Shannon entropy. The minimal requirement to fully reconstruct the message is that the sum of rate $R_0(\epsilon)$ over all bits is at least the size of the message.

We will discuss sequential decoding for reconstruction purpose, which turns out to work close to this limit. Theoretical analysis and tests of accompanied implementation show that the statistical behavior of constructed tree can be approximated by Pareto distribution with coefficient such that the sum of $R_c(\epsilon) = 1 - h_1/(1+c)(\epsilon)$ is the size of the message, where $h_u(\epsilon) = (\epsilon^u + (1-\epsilon)^u)/(1-u)$ is $u$-th Renyi entropy.

This possibility could also improve throughput and reduce costs (hardware and energy consumption) of various networks by shifting error correction (of the payload) from applied for each connection, to a single combined correction of all received damaged packets - reconstructing the message from all available scraps of information.

Keywords: fountain codes, error correction, sequential decoding, Renyi entropy, cryptography
1 Introduction

Many communication settings require dividing a potentially large message (payload) into relatively small blocks of data, which will be referred here as packets. These packets are often damaged or even lost on the way. Both issues could be handled by allowing for retransmission of uncertain or missing parts of data. However, it would require costly bidirectional communication, waiting for packets before classifying them as lost (automatic repeat request after timeout). Hence, there is a strong need for methods not requiring feedback, schematically presented in Fig. 1.

The issue of missing packets can be handled by various erasure channel methods, for example by the fact that coefficients of degree \( d \) polynomial can be obtained from values in any \( d + 1 \) points in Reed-Solomon codes [1]. A faster way, pioneered in LT codes by Michael Luby [2] in 2002, is constructing the packets as XOR of some different subsets of data blocks. This approach is currently referred as Fountain Codes (FC) or rateless erasure codes and was improved to linear encoding and decoding time in Raptor Codes [3]. There are considered various applications like relay networks [4], sensor networks [5], radio networks [6], networked storage [7] or watermarking [8].

If a packet was not lost, it is often received damaged. To handle this issue without retransmission there is used some forward error correction (FEC) method: attaching some redundancy, which will be used by decoder to repair the packet. The applied redundancy

![Figure 1: Schematic picture of different possibilities of splitting the informational content of the message. 1) Simple split requires all undamaged packets. 2) For Fountain Codes, any large enough subset of undamaged packets is sufficient. 3) While the set of packets is already redundant, we can add additional redundancy to separately protect each packet – some turn out overprotected, some insufficiently. 4) JRC allows to extract the actual informational content (green area) from all obtained packets. Intuitively, we need to gather sufficient amount of informational content: such that the sum of \( 1 - h(\epsilon) \) over all bits exceeds the size of the message, where \( \epsilon \) is bit-flip probability.](image)
level defines some stochastic threshold for damage, determining if the packet can be fully repaired. Fountain Codes do not tolerate damaged packets, so those exceeding the damage threshold need to be discarded - their entire informational content is being lost. As the threshold has stochastic nature, this all-or-nothing means inefficient use of the channel: some packets turn out to be overprotected - have used an unnecessarily large amount of redundancy, some packets insufficiently - loosing all the content.

Additionally, in many scenarios the sender has inaccurate or no a priori knowledge of the final damage level, hence a strong overprotection need to be used. For example

- while protecting a storage medium, the final damage level depends on age, conditions and random accidents,
- while sending a packet through a network, there is usually no knowledge of the route it will take,
- while embedding a watermaring, there is no knowledge of the final capturing conditions,
- the noise level can vary too quickly for adaptive change of redundancy level, for example in acoustic or radio communication (e.g. underwater).

This damage level can be usually estimated a posteriori - correspondingly: basing on the history of this storage medium, on the actual route of a given packet, on the parameters of captured image, on more recent evaluation of environmental parameters. Moreover, the applied redundancy could allow to extract the actual damage level from the received packets. Therefore, it would be beneficial to allow to shift the need of knowledge of damage level from a priori to a posteriori.

This paper introduces and discusses Joint Reconstruction Codes (JRC) which, offering the same rates, requires only a posteriori knowledge of the damage level thanks to combining both processes: of error correction and of reconstruction from multiple packets. It prepares packets to be treated simultaneously as the message (payload) and FEC redundancy. The decoder searches the space of promising candidates for the message sequence to agree with all received packets accordingly to their a posteriori damage level - individual trust level for each packet.

For simplicity we will consider only Binary Symmetric Channel (BSC) here: that each bit has independent $\epsilon$ probability of being flipped. Theoretical rate limit in this case is $R_0(\epsilon) := 1 - h(\epsilon)$, where $h(p) = -p \log(p) - (1 - p) \log(1 - p)$ is the Shannon entropy. So the minimal requirement for reconstructing a message is that the sum of $1 - h(\epsilon)$ over all received bits is at least the size of the message. It is schematically presented in Fig. 1 by green bars having $1 - h(\epsilon)$ size of the packet - the minimal requirement for reconstruction is that the green bars sum to the length of the message.

The practical error correction methods usually work below this theoretical rate. We will discuss enhancement of sequential decoding, which is related to so called cutoff rate [9]. We will use generalized family of rates described by $c \geq 0$ parameter (Pareto coefficient) as in Fig. 2:

$$R_c(\epsilon) = 1 - h_1/(1+c)(\epsilon)$$

where

$$h_u(\epsilon) = \frac{\log(u \epsilon^u + (1 - \epsilon)^u)}{1 - u}$$

(1)
1 INTRODUCTION

Figure 2: Rate: informational content per bit assuming Binary Symmetric Channel. Renyi entropy $h_u(\epsilon) = \lg(\epsilon^u + (1 - \epsilon)^u)/(1 - u)$ becomes Shannon entropy for $u \to 1$ ($c \to 0$). The $c = 0$ rate is the theoretical bound. The remaining cases allow to find Pareto coefficient $c$ describing correction process: such that sum of $R_c(\epsilon)$ over all bits is the size of the message.

is $u$-th Renyi entropy $[10]$, $\lg \equiv \log_2$. Additionally, the we need to define $h_1$ as the limit:

$$h_1(\epsilon) \equiv h(\epsilon) = -\epsilon \lg(\epsilon) - (1 - \epsilon) \lg(1 - \epsilon)$$

is the Shannon entropy. (2)

Finally we have the mentioned two special cases:

$$R_0(\epsilon) = 1 - h(\epsilon) \quad R_1(\epsilon) = 2 \lg(\sqrt{\epsilon} + \sqrt{1 - \epsilon}) = \lg(1 + 2\sqrt{\epsilon(1 - \epsilon)})$$

(3)

The $R_0$ is theoretical rate limit for this channel. The $R_1$ is cutoff rate for sequential decoding [9]: below this rate, sequential decoding of infinite message would require a finite width - average number of steps per bit block. Generally, $c$ is the coefficient of Pareto distribution describing the statistics of correction process: increasing twice the maximal number of considered steps, probability of failure drops asymptotically $2^c$ times. Thanks of considering finite messages and some improvements to sequential decoding: large internal state (64 bit) and bidirectional correction, much larger rates than $c = 1$ cutoff rate can be obtained [11], e.g. for $c = 1/2$ still got complete correction in the considered cases.

Finally, the $c$ value describing correction process is such that the sum of $R_c(\epsilon)$ over all bits is the size of the message. The larger $c$, the easier/cheaper the correction (statistically). So from JRC perspective we should choose some boundary value of $c_{\min}$ and wait until the sum of $R_{c_{\min}}(\epsilon)$ over received packets exceeds the message size. Then decoder can try to perform the correction and eventually wait for another packet(s) to try again if it failed to reconstruct within assumed resource limit (time and memory). For unidirectional sequential decoding (implemented) this $c_{\min}$ boundary can be chosen in $\approx [0.5, 1]$ range, depending on resource limit. Bidirectional sequential decoding effectively allows to halve this value.

Let us briefly look at some applications of this possibility of removing the need of a priori knowledge of damage level, like protecting storage media. For example imagine a thousand DVD copies of a movie. While time passes, all of them will degrade, finally exceeding included protection level - making all of them useless. JRC approach would allow to still reconstruct the original content from some number of badly damaged copies, where the required number depends on damage level.

Other family of applications is improving efficiency and reducing energy consumption of various networks by replacing error correction applied by every node, with a single reconstruction+correction applied by the receiver only (of the payload - some small packet’s
header need to be frequently corrected). Decreasing energy consumption of network nodes is especially important for battery operated, e.g. sensor networks.

For watermarking applications, we could divide a message into small packets - the receiver should be able to reconstruct the message even if some of them are missing (e.g. part of a picture or frames of a movie) and the remaining packets are damaged - in a way unknown to the sender.

This approach could be also used when there are no missing packets (some additional optimization could be applied in this case). For example imagine acoustic or radio communication while rapidly varying environmental conditions - the receiver can likely receive all the packets, and he usually have more information about the actual damage level than the sender, making JRC beneficial.

The article is constructed in the following way. Section 2 formulates the problem, introduces encoding scheme and separate decoding procedures in two cases: for undamaged packets only and for the general case (with included error correction). Section 3 discusses the undamaged case - both theoretically and comparing with experimental tests. It starts with straightforward decoding: when we need to consider only a single candidate. Then a general undamaged case is discussed, where it turns out that we need to consider only a small number of candidates per step (on average $\leq 2$). Section 4 contains theoretical analysis and results of experiments for the general damaged case, where the number of candidates per step is approximately described by Pareto distribution with coefficient found using Renyi entropy. Finally, Section 5 summarizes the new possibilities and discusses some further research questions and improvement opportunities.

2 Joint Reconstruction Codes (JRC)

We will now formulate the problem and introduce coding/decoding methods. The details can be found in accompanied implementation [12].

![Diagram](image)

Figure 3: Two approaches of combining Fountain Codes (FC) with error correction (two "X"-marked packets have been lost) for $m = 3$. Left: correct each packet independently. Right: JRC tries to reconstruct succeeding bits of the message from bits on corresponding position of simultaneously all packets.
2.1 Encoding

Let us start with the Fountain Code (FC) situation. Assume the size of the message is $Nl$ and we are receiving $m$ packets of $l$ bits. So among a larger set of packets, we need at least some $m \geq N$ (undamaged) packets to reconstruct the message. We will now allow the messages to be damaged, as depicted in Fig. 3. For simplicity there will be assumed binary symmetric channel (BSC): packet $i \in [1, m]$ has $\epsilon_i$ probability of its bits being flipped. Applying forward error correction correspondingly to each channel, the minimal requirement for reconstruction is $\sum_i (1 - h(\epsilon_i)) \geq N$. We will get the same bound for JRC.

Let us divide the message into length $N$ bit blocks, each block corresponds to a single bit in every packets (vertical lines in Fig. 3). Specifically, the encoding procedure has $l$ steps. In $k$-th encoding step ($k \in \{0, \ldots, l-1\}$) there are used $\{Nk, \ldots, N(k+1)-1\}$ bits of the message (bit block $x$), to produce $k$-th bit of every packets. There is required an internal state of encoder to connect redundancy of succeeding blocks. The current implementation uses 64 bit state for this purpose, producing $k$-th bit of $i$-th packet as $i$-th bit of the state. The number of produced packets is limited to 64 this way, the actually received packets correspond to some subset of these 64 bits. The internal state needs to be modified accordingly to currently encoded $N$ bit block $x$. There is used a pseudorandomly chosen transition function $f : \{0, \ldots, 2^N - 1\} \rightarrow \{0, \ldots, 2^{64} - 1\}$ for this purpose. The state transition is cyclic shift of (state XOR $f[x]$). The encoding procedure is schematically presented as Algorithm 1, example of its application is presented in Fig. 4. Observe that using a Pseudorandom Number Generator initialized with a cryptographic key to choose the $f$ function, we could include encryption in such encoding.

Finally encoding procedure is: set state as some arbitrary initial state (known to receiver), then perform encoding step for blocks 0 to $l-1$. The final state: final state would be beneficial for decoder for final verification of unidirectional decoding, or is necessary for bidirectional decoding. It can be included in the header of packet. However, unidirectional correction can be performed without it, at cost of probable damage of some last bits of the message.

**Algorithm 1** Encoding procedure

```plaintext
Require: state = initial state
for k = 0 to l - 1 do
    set x as k-th N bit block to encode: \{Nk, ..., N(k+1)-1\} bits of the message
    state = state XOR f[x] \{XOR with corresponding transition function\}
    for all i set k-th bit of packet i as i-th bit of state
    state = state >> 1 \{cyclic shift by one position\}
end for
final state = state \{it can be useful for decoding\}
```

2.2 Decoding in the undamaged case - no error correction

The decoding process tries to find a message leading to all the received packets by expanding a tree of promising candidates. Depth $k$ node of this tree represents a candidate of length $Nk$ prefix of the message, and $k$ length prefixes of all received packets. Expanding a node means creating its children as corresponding to $N$ bit block longer prefix of the message.
Figure 4: Example of encoding process for 5 bit state and 5 packets, \( N = 2 \) size bit blocks, \( m = 3 \) received packets and \( f[00] = 01100, f[01] = 10010, f[10] = 01010, f[11] \) = 10110 transition function, chosen in a pseudorandom way. \( \oplus \) denotes XOR, "\( >\)" denotes cyclic shift right by 1 position of the 5 bit state.

In the undamaged case we can consider only candidates having full agreement with the received packets, the remaining expansions are not considered. In this case, a natural approach is to consider a list of all candidates up to a given position. The list for succeeding position is obtained by expanding all candidates from the previous list by a single step, considering only those being in agreement with the corresponding bit of all received packets.

For performance reasons, this method should have prepared a list of allowed single step expansions for all cases. Denote by \( data[k] \) the \( m \) bit sequence of \( k \)-th bit of all received packets. Denote by \( extract(state) \) as \( m \) bit sequence of bits of \( state \) on positions corresponding to received packets. Now a node can be expanded by sequence \( x \) if \( data[k] = extract(f[x] XOR state) \), or in other words if \( extract(f[x]) = extract(state) XOR data[k] \). So the current situation can be described by one of \( 2^m \) values \( z \):

\[
    z = extract(state) XOR data[k]
\]

**Algorithm 2** Sequential decoding for undamaged case(schematic)

| Require: current_list = {root} \( \) (initial state, \( k = 0 \) position before the first bit block) |
| Require: candidates[z] is a list of all \( x \) such that \( extract(f[x]) = z \) |
| for \( k = 0 \) to \( l - 1 \) do |
| empty new_list |
| for every node in current_list do |
| append candidates[extract(node.state) XOR data[k]] to new_list |
| end for |
| current_list = new_list |
| end for |
| trace back any coding from the final new_list (with state=final_state if available) |

For each value \( z \) we can have prepared a list of \( x \) in agreement in a table: candidates[z]. As \( m \geq N \), the expected size of each of such lists is \( \leq 1 \), so the total size of candidates table
is $O(2^m)$. The decoding method is schematically presented as Algorithm 2. Its performance is discussed in Section 3.

2.3 Decoding in the general case - with included error correction

In the general case we allow for damaged packets: reconstruct tolerating some disagreement with the received packets. The larger assumed (a posteriori) damage level ($\epsilon_i$), the higher tolerance for disagreeing bits. Considering all candidates up to a given position would be completely impractical here as their number would grow exponentially. Instead, in each step we will find and expand the most promising node, as the one maximizing weight $W$, which is logarithm of the probability of being the proper node, obtained using Bayesian analysis. As discussed in [11], this probability is a product of probability of assumed error vector and one over probability of accidental agreement of incorrect node.

Assuming an incorrect node, the probability of accidental agreement of some of 2$^N$ its possible expansion with $m$ bits of received packets is $2^N / 2^m$. Let us denote $k$ step error vector for given message candidate by $\mathbf{E}^k = \{E^k_i\}$: $E^k_i = 0$ if $k$-th bit of $i$-th packet agrees, 1 otherwise. Now assuming $i$-th packet came through $\epsilon_i$ BSC, probability of $\mathbf{E}^k$ error vector is:

$$\Pr(\mathbf{E}^k) = \prod_{i=1}^{m} p(\epsilon_i, E^k_i)$$

where $p(\epsilon, 0) := 1 - \epsilon$ , $p(\epsilon, 1) := \epsilon$ (4)

We can finally write formula for weight we want maximize:

$$W = \sum_k W(\mathbf{E}^k)$$

$$W(\mathbf{E}^k) = m - N + \sum_{i=1}^{m} \log(p(\epsilon_i, E^k_i))$$ (5)

In a given moment, the most probable node to expand is the one having the largest weight $W$. The $m - N$ term favors longer expansions, the error vector term contains a penalty for disagreement with the received damaged packets. The weight of expansion of a node is the weight of its parent plus $W(\mathbf{E}^k)$ for the currently considered expansion, describing disagreement with $k$-th bit of the packets.

In practical applications, we need to quickly find the best candidates for expansion. As for the undamaged case, the current situation is sufficiently described by $z = \text{extract} \ (\text{state} \ \text{XOR} \ y)$. So this time $\text{candidates}[z]$ lookup table should contain $2^N$ bit block candidates $x$ for the message, sorted by the $W(z \ \text{XOR} \ \text{extract}(f[x]))$. The final decoding method is schematically presented as Algorithm 3, detailed implementation is in [12], its performance analysis is discussed in Section 4.

**Algorithm 3** Sequential decoding for general case (schematic)

**Require:** empty *heap* of nodes to consider

**Require:** candidates[$z$] is a list of 2$^N$ blocks $x$ sorted by $W(z \ \text{XOR} \ \text{extract}(f[x]))$

insert root to *heap: initial state* and $k = 0$ position before the first block

repeat
node = getMax(*heap*) \{retrieve maximal weight node from *heap*\}
if applicable, add next element from candidates of parent of node to *heap*
add first of candidates of node to *heap*
until reached final position (and if available: state=final state)
3 Undamaged packets case

Let us now discuss the situation with only undamaged packets - analogous to Fountain Codes, no error correction included. We can use simpler decoding Algorithm 2 in this case: consider all candidates up to a given position and successively shift this position.

3.1 Straightforward decoding

Let us look at \( \{\text{extract}(f[x])\}_{x=0,\ldots,2^N-1} \) length \( m \) sequences: containing bits of \( f[x] \) corresponding to all received packets. Observe that if all these sequences turn out different, the block of \( k \)-th bit of the received packets (\( \text{data}[k] \)) immediately determine the unique \( x \) to decode: candidates[extract(state)] XOR data[\( k \)] is a single value. We will call this case as straightforward decoding case: we can directly determine the only possible candidate, move to succeeding position and so on. The number of decoding steps is exactly the number of encoding steps in this case.

Assuming a completely random transition function \( f \) (\( \Pr(0) = \Pr(1) = 1/2 \)), we can find probability of such straightforward decoding: that \( 2^N \) random length \( m \geq N \) sequences will be different. Table 1 contains some its numerical values.

\[
\Pr(\text{straightforward decoding}) = \frac{(2^m)!}{(2^m - 2^N)!} \quad (6)
\]

| \( N/m \) | 1   | 2      | 3   | 4   | 5   | 6   | 7   | 8   |
|----------|-----|--------|-----|-----|-----|-----|-----|-----|
| 1        | 0.5 | 0.75   | 0.875 | 0.9375 | 0.96875 | 0.98436 | 0.99219 | 0.99609 |
| 2        | -   | 0.09375 | 0.41016 | 0.66650 | 0.823059 | 0.90891 | 0.953794 | 0.97673 |
| 3        | -   | -      | 0.00240 | 0.12082 | 0.38572 | 0.634028 | 0.79999 | 0.89542 |
| 4        | -   | -      | -      | 1.1 \cdot 10^{-6} | 0.01040 | 0.12901 | 0.37613 | 0.61971 |
| 5        | -   | -      | -      | -      | 1.8 \cdot 10^{-13} | 7.6 \cdot 10^{-5} | 0.01442 | 0.13236 |
| 6        | -   | -      | -      | -      | -      | 3 \cdot 10^{-27} | 4.2 \cdot 10^{-9} | 0.00018 |
| 7        | -   | -      | -      | -      | -      | -      | 7 \cdot 10^{-55} | 1 \cdot 10^{-17} |
| 8        | -   | -      | -      | -      | -      | -      | -      | 3 \cdot 10^{-110} |

We can see that assuming a completely randomly chosen transition function \( f \), for some parameters we can get straightforward decoding with a reasonable probability. Usually we have some (statistical) influence on the subset of packets used for reconstruction, for example by the order of sending: the receiver is more likely to use some number of the first sent packets. In this case we can prepare them to increase the probability of straightforward decoding: by choosing \( f \) to make that extract(\( f[x] \)) are far from each other.

3.2 General undamaged case

Let us define width characterization of given decoding process as the average number of candidates considered per position.
The number of encoding steps for \( lN \) bit message is \( l \). The number of decoding steps is the number of considered candidates: nodes of reconstruction tree.

For straightforward decoding \( width = 1 \), what is the lower bound. We are interested in understanding probability distribution of \( width \) for given reconstruction parameters. We will now explain that we should expect approximately Gaussian distribution for the undamaged \( m > N \) case, what is confirmed by tests. For damaged case: with included error correction, we get approximately Pareto distribution of \( width \), what will be discussed in Section 4.

Figure 5 shows \( widths \) for \( m = N + 1 \) cases and \( N = 1, \ldots, 8 \). It was obtained by performing 1000 reconstructions for random conditions (message and subset of packets) using Algorithm 3 and then sorting obtained \( widths \). If we would switch both axes of this plot, we would get empirical approximation of cumulative distribution function for \( width \). In its left panel (low \( N \)) we can see some \( width = 1 \) lines corresponding to straightforward decoding - up to approximately 0.75, 0.41, 0.12, 0.01 of 1000 for correspondingly \( N = 1, 2, 3, 4 \), as we can read from Table 1.

For a larger \( N \) we can see a shape resembling cumulative distribution function of Gaussian distribution. Let us try to find this behavior assuming some randomness. Imagine that in a given step we have considered \( i \) candidates. They have \( 2^N i \) possible expansions - one of them is the proper message, the remaining have \( 2^{-m} \) probability of accidentally agreeing with the \( m \) bits from the received packets. Finally, the probability that there will be \( j \) candidates to consider in the next position is

\[
P_{ij} = \binom{2^N i - 1}{j - 1} 2^{-m(j-1)}(1 - 2^{-m})^{2^N i - j}
\]

This stochastic matrix (of infinite size) allows to find the stationary probability distribution for the number of candidates to consider: such that \( p_i = \sum_j p_j P_{ij} \), where \( p_i \) is the probability that there will be \( i \) candidates in a given position. This \( \{p_i\}_{i=1}^\infty \) distribution tends to 0 for

![Figure 5: Experimental results for Algorithm 3. N = 1, 2, 3, 4, 5, 6, 7, 8 required packets and received m = N + 1 undamaged packets. The graphs show sorted widths for 1000 trials for length 1000B messages (l = \( \lceil 8000/N \rceil \) steps).](image)
For $m = N + 1$ case it stabilizes for large $N$ on approximate distribution:
\[
\{0.41884, 0.32221, 0.15680, 0.06446, 0.02438, 0.00875, 0.00303, 0.00102, 0.00034, 0.00011, \ldots \}
\]
with average value 2 and standard deviation 3.28. For $m = N + 2$ case the asymptotic distribution is approximately:
\[
\{0.72383, 0.22726, 0.04175, 0.00620, 0.00084, 0.00011, 0.00001, \ldots \}
\]
with average value $4/3$ and standard deviation 2.58.

Finally, for the $m = N + 1$ case and $N \gg 4$, the width of Algorithm 2 should be approximately a Gaussian distribution with expected value 2: 1 proper candidate plus on average 1 improper one per position. However, Figure 5 clearly shows expected value 1.5 - this difference is caused by using Algorithm 3 for these results. This algorithm considers simultaneously candidates of various lengths, favoring the longest one. So after considering the proper candidate, it will not consider the remaining candidates on a given position. The proper candidate is on average in the middle, so the expected width for Algorithm 3 should be 1 (proper) + 0.5 (improper) = 1.5 here, what agrees with the Figure 5.

One could wrongly conclude that it is better to use Algorithm 3 for the undamaged case to reduce the linear coefficient e.g. from 2 to 1.5 here. However, this algorithm requires finding the best node for each step, making it more costly than Algorithm 2.

### 3.3 Conclusions and comparison with Fountain Codes

In this section we have discussed proposed coding approach as a direct replacement for Fountain Codes: for undamaged packets. With some probability $m = N$ packets are sufficient to fully reconstruct the message, especially for small $N$. For the $m = N + 1$ case: one excess packet, the decoding is very cheap: requires on average twice more steps than encoding. Probability of failure is practically zero.

In contrast, Fountain Codes have relatively large probability of failure for a small number of excess packets (the used random matrix is not invertible), like $\approx 50\%$ for the $m = N + 1$ case [13]. However, they can operate on arbitrarily large number of packets. Discussed Algorithm 2 requires the candidates[] table for performance reasons, which has $O(2^m)$ size, bounding the number of packets we can use for reconstruction.

### 4 General damaged case

We will now discuss the performance of the general case: with allowed damaged packets. This time decoding is much more costly due to included error correction. While for undamaged case the width had approximately Gaussian distribution with a small expected value ($\approx 2$ for $m = N + 1$ case), the width for damaged case is approximately described by Pareto distribution with parameter $c \geq 0$: increasing twice the width limit decreases probability of failure approximately $2^c$ times. The $c = 0$ case is the Shannon theoretical rate limit. The $c = 1$ case is called cutoff rate: the maximal rate with finite asymptotic width ($\int_0^\infty x^{-c}dx = \infty$ for $c \leq 1$). We will now find $c$ for a general case. Derivation for simpler case: single BSC and tests results can be found in [14].
4.1 Finding the Pareto coefficient $c$

Let us remind weights found in Section 2 quantitatively describing how likely we will tolerate bit-flips depending on damage level of each packets (that $i$-th packet came through $\epsilon_i$ BSC). Denote error vector as $E^k = \{E^k_i\}_{i=1..m}$, where $E^k_i = 0$ if $k$-th bit of $i$-th packet agrees, $E^k_i = 1$ otherwise. Now the weight of $k$-th block of candidate is $W(E^k)$:

$$W(E) = m - N + \sum_{i=1}^{m} \lg(p(\epsilon_i, E_i))$$

where $p(\epsilon, 0) := 1 - \epsilon$, $p(\epsilon, 1) := \epsilon$.

The total weight of a candidate is the sum of its $W(E^k)$. In every step of decoding Algorithm 3 we choose the looking most probable node of the tree (candidate): having the largest weight, and try to expand it.

The Shannon rate limit is $\sum_{i=1}^{m} (1 - h(\epsilon_i)) \geq N$, what is equivalent to $\sum_{E} \Pr(E)W(E) \geq 0$. In other words, for rates below the Shannon limit, the weight of the proper candidates is on average growing. In contrast, the weights of improper candidates is on average decreasing, allowing to cut these branches of reconstruction tree.

The situation is depicted in Fig. 6 - its dots correspond to length $N$ bit blocks of the message. Let us imagine the (unknown) proper reconstruction. As long as there are no errors, its weight will grow ($W(E) > 0$), hence it will remain the favorite candidate of sequential decoding. However, its weight can drop while errors appear ($W(E) < 0$). In this case, the decoder will try different (improper) candidates, until exploring all possibilities having higher weight than the weight for the proper reconstruction (triangles in the figure). So the crucial is weight drop $w$ parameter (arrows in the figure): the maximal weight up to the considered position minus the current weight. Usually we should have $w = 0$: given proper node has the highest weight so far. However, larger $w$ has some exponentially decreasing probability of appearing, and requires considering subtrees of improper candidates of size growing exponentially with $w$.

We will now find these two behaviors. Let us start with finding the expect number of improper nodes to expand for a given weight drop. Define

$$U(w) := \text{the expected number of improper nodes in subtree with } w \text{ weight drop}$$

If a given node is created, we potentially need to expand all of its $2^N$ children. Let us assume that each of them corresponds to a completely random $E = \{E_i\}$ vector:

![Figure 6: Schematic picture of the proper reconstruction we would like to find. After an error, we expand improper candidates until reaching the weight of the proper one. We need to find the expected size ($U(w)$) and probability distribution ($V(w)$) for these subtrees of improper candidates.](image)
\[ \Pr(E) = \frac{1}{2^m}. \]

Finally, the expected number of nodes to expand for each of \(2^N\) possibilities is \(\sum_{E\in\{0,1\}^m} U(w + W(E))/2^m\), getting functional equation:

\[
U(w) = \begin{cases} 
1 + 2^N \sum_{E\in\{0,1\}^m} U(w + W(E))/2^m & \text{for } w \geq 0 \\
0 & \text{for } w < 0 
\end{cases} \tag{8}
\]

Large \(w\) behavior is crucial here - it corresponds to rare large subtrees of improper candidates. In this case we can neglect the ”1+” term and the remaining linear functional equation should have asymptotic behavior of form:

\[
U(w) \propto 2^{uw} \quad \text{for } w \to \infty \quad \text{and some } u > 0
\]

Inserting this assumption to the functional equation we get:

\[
2^{uw} = 2^{N-m} \sum_{E\in\{0,1\}^m} 2^{u(w+m-N+\sum_i \log(p(\epsilon_i, E_i)))}
\]

\[
2^{N-m} = \sum_{E\in\{0,1\}^m} \prod_i p(\epsilon_i, E_i)^u = \prod_{i=1}^m \left( \epsilon_i^u + (1 - \epsilon_i)^u \right)
\]

\[
N = m - \sum_{i=1}^m \frac{\log(\epsilon_i^u + (1 - \epsilon_i)^u)}{1-u} = \sum_{i=1}^m 1 - h_u(\epsilon_i) \tag{9}
\]

where \(h_u(\epsilon) = \log(\epsilon^u + (1-\epsilon)^u) / (1-u)\) is Renyi entropy.

Let us now find the probability distribution of appearing values of \(w\). Define

\[
V(w) := \text{probability that the weight on the correct path will drop by at most } w
\]

Comparing situation in succeeding positions and using BSC assumption: \(\Pr(E) = \prod_{i=1}^m p(\epsilon_i, E_i)\), we get relation:

\[
V(w) = \begin{cases} 
\sum_{E\in\{0,1\}^m} \prod_{i=1}^m p(\epsilon_i, E_i) \cdot V(w + W(E)) & \text{for } w \geq 0 \\
0 & \text{for } w < 0 
\end{cases} \tag{10}
\]

For \(w < 0\), \(V(w) = 0\). As discussed, \(V(0) > 0\) is the probability the a proper node will have the highest weight so far. From definition, \(\lim_{w\to\infty} V(w) = 1\). Finally, we can assume that the functional equation has solution of form:

\[
1 - V(w) \propto 2^{-vw} \quad \text{for } w \to \infty \quad \text{and some } v > 0
\]

\[
2^{-vw} = \sum_{E\in\{0,1\}^m} \left( \prod_i p(\epsilon_i, E_i) \right) \cdot 2^{-v(w+m-N+\sum_i \log(p(\epsilon_i, E_i)))}
\]

\[
2^{v(m-N)} = \sum_{E\in\{0,1\}^m} \prod_i p(\epsilon_i, E_i)^{1-v} = \prod_{i=1}^m \left( \epsilon_i^{1-v} + (1 - \epsilon_i)^{1-v} \right)
\]

\[
N = m - \sum_{i=1}^m \frac{\log(\epsilon_i^{1-v} + (1 - \epsilon_i)^{1-v})}{v} = \sum_{i=1}^m 1 - h_{1-v}(\epsilon_i) \tag{11}
\]
comparing with formula (9) for \( u \), we see that \( v = 1 - u \).

Having the exponent coefficients of \( U(w) \) and \( V(w) \) functions from (9) and (11), we can find the asymptotic behavior of probability that some number of steps would be exceeded for a single position node. Assume \( U(w) \approx c_u 2^{uw} \), \( V(w) \approx 1 - c_v 2^{-vw} \) for some unknown coefficients \( c_u \), \( c_v \). Now the asymptotic probability that the number of nodes of subtree of wrong correction for a given position will exceed some number of steps \( s \) is:

\[
\Pr(\# \text{ nodes } > s) \approx \Pr \left( w > \frac{\log(s/c_u)}{u} \right) \approx 1 - c_v 2^{-\frac{s}{u} \log(s/c_u)} = 1 - c_v \left( \frac{s}{c_u} \right)^{-v/u}
\]

\[
\Pr(\# \text{ nodes in improper subtree } > s) \approx 1 - c_p s^{-c}
\]

where \( c := v/u = 1/u - 1 \) and \( c_p = c_v/c_u \). So asymptotically we have obtained the Pareto probability distribution with exponent which can be found analytically. Finally the Pareto coefficient is \( c \) (\( u = 1/(1+c) \)) if the sum over all packets is the number of required packets: \( N = \sum_{i=1}^{m} 1 - h_1/(1+c) (\epsilon_i) \). In other words, the sum of \( R_c(\epsilon) = 1 - h_1/(1+c)(\epsilon) \) over all received bits should be the number of bits of the message.

These were considerations for unidirectional decoding: in direction of encoding, starting from the initial state. As discussed and tested in [11] for single BSC, we could also perform bidirectional correction: simultaneously start backward direction sequential decoding from the final state. Failure of decoding is caused by critical error concentration (CEC): rare event requiring to consider very large number of improper candidates (large weight drop \( w \)) - exceeding assumed resource limit. While single CEC prevents successful reconstruction for unidirectional decoding, bidirectional decoding can perform nearly complete reconstruction in this case (and mark the uncertain part). Two CECs are required to essentially cripple the process in this case: probability of such event is approximately square of probability of a single CEC. Finally, Pareto coefficient \( c \) practically doubles in this case: \( c = 1/2 \) bidirectional correction has similar behavior as \( c = 1 \) unidirectional.

### 4.2 Experimental tests

Extensive tests for single BSC, including bidirectional correction, can be found in [11]. We will now discuss two experiments for the general JRC case relying on using multiple packets. Implementation [12] were used for the tests (unidirectional), both tests use 1000B message.

The first experiment, presented in Figure 7, uses \( N - 1 \) undamaged packets and two damaged packets: of \( \epsilon = 0.04 \) and 0.05, chosen to get \( c \approx 1 \) (cutoff rate, \( \sum_i R_1(\epsilon_i) \approx N \)). Increasing \( N \) does not affect \( c \), but the graphs show some dependence on \( N \): smaller \( N \) case has a bad behaving tail, stabilized thanks to using more nodes.

The second example, presented in Figure 8 tests the exploitation of tiny informational content of highly damaged packets: of \( \epsilon = 0.2 \). This level of damage is difficult to handle with standard error correction, while here we see that they can be used as a replacement for a missing undamaged packet, and increase of their number essentially improves the correction process. The graphs show that the Pareto assumption is not in perfect agreement with experiment, however the found coefficients describe well the general behavior.
4 GENERAL DAMAGED CASE

Figure 7: Experimental results (solid lines) and Pareto approximation (dashed line) for $N = 5, 6, 7, 8$ required packets, $N - 1$ received undamaged and two packets having damage level chosen to obtain $c \approx 1$ Pareto coefficient ($\epsilon = 0.05$ and 0.04). While they have the same $c$, larger $N$ allows to stabilize the correction process. Left: sorted average width from 1000 trials. Right: the same plot in log-log scale for number of trial rescaled to $[0, 1]$ range.

Figure 8: Experimental results (solid lines) and Pareto approximation (dashed lines) for $N = 3$ required packets with 2 undamaged and $d = 6, 7, 8, 9$ badly damaged packets: with $\epsilon = 0.2$ probability of bit-flip. Left: sorted average width from 1000 trials. Right: the same plot in log-log scale for number of trial rescaled to $[0, 1]$ range.

4.3 Summary

Let us summarize the theoretical part: obtaining $m$ packets with correspondingly $\{\epsilon_i\}_{i=1..m}$ bit-flip probability, we should find $c$ such that $\hat{N} = \sum_{i=1}^{m} R_c(\epsilon_i) = \sum_{i=1}^{m} 1 - h_{1/(1+c)}(\epsilon_i)$.

As $h_u(0) = 0$ and generally $h \geq 0$, if there is $N$ or more undamaged packets, no such $c$ can be found - we have the undamaged case with Gaussian distribution of weights (if receiving anything more than $N$ undamaged packets).

Otherwise, we have approximately $c$ Pareto distribution of weight if $c > 0$. If $c < 0$, there is definitely not sufficient information for reconstruction - we have to wait for more packets. For $0 \leq c \leq 1$ the (average) weight for infinite packets would be infinite ($\int_{0}^{\infty} x^{-c}dx = \infty$ for $c \leq 1$). However, finite length packets can be still handled in this range. Finally, the
receiver should wait until reaching some arbitrarily chosen $c_{\text{min}}$ value:

$$\sum_{i=1}^{m} R_{c_{\text{min}}} (\epsilon_i) \geq N,$$

(13)

then perform the first trial of reconstruction. If assumed resource limit (time and memory) was not sufficient, it should wait for another packet (or more), then try reconstruction one more time and so on. The succeeding trials can use partial reconstruction from the previous trials.

The question is what minimal value ($c_{\text{min}}$) should we choose before the first trial. Optimally choosing this parameter is a difficult question, generally $c_{\text{min}} \in [1/2, 1]$ is suggested for unidirectional correction, twice smaller for bidirectional.

5 Conclusions and further perspectives

The article introduced and discussed Joint Reconstruction Codes (JRC) as enhancement of Fountain Codes (FC) concept. JRC allows to combine together two processes: of reconstruction from some subset of packets and of error correction. Standard approaches is using FC with packets protected by Forward Error Correction in such case, but it requires a priori knowledge of damage level, unavailable in many scenarios. In contrast, JRC allows to operate with the same rates having only a posteriori knowledge of damage levels. It searches the space of candidates of the message to get agreement with the received packets, tolerating inconsistencies accordingly to the individual damage level of each packet, estimated a posteriori. Thanks of it, the sender needs only to ensure a high enough statistics of packets for the receiver, do not have to care about the protection level for each packet - they are simultaneously payload and redundancy. Presented theoretical analysis allows to calculate Pareto coefficient using Renyi entropy, approximately describing the sequential decoding process.

Some examples of applications:

- protecting a data storage: allowing to reconstruct the content from some a priori unknown subset of storage media, having a history dependent actual damage levels,

- for reducing hardware cost and energy consumption of various networks by replacing correction applied by every node, into a single reconstruction performed by the receiver,

- for more robust watermarking: requiring some subset of frames or parts of a picture, with capture conditions unknown to the sender,

- for communication with rapidly varying condition, where the receiver has usually a better (a posteriori) information about the actual damage level.

Finally, the undamaged case shows that the presented approach also has advantages as a direct replacement for FC. For example in the case of one excess packet ($m = N + 1$), decoding needs to consider on average only twice more steps then encoding and practically always succeeds. In contrast, FC often fails in this case (with $\approx 50\%$ probability).
This paper only introduces to this topic - there have remained plenty of research questions and possibilities for improvements and optimizations depending on specific applications.

The implemented and analyzed decoding is currently unidirectional, what could be essentially improved by using bidirectional sequential decoding, effectively doubling the Pareto coefficient [11]. Another line of development is including more sophisticated types of errors in the considered space of candidates/corrections - the discussed approach can even handle the synchronization errors, like deletion channel [14]. It might be also beneficial to compare the discussed sequential decoding approach with different ones for similar purpose, like Low Density Parity Check [15].

Additional improvement of reconstruction efficiency can be made by optimizing the choice of transition function $f$, for example including the order of sent packets into consideration, especially when the sender is likely to receive most or all of the packets. Specifically, it should be chosen such that the set \{extract($f[x]$)\}$_x$ contains unique sequences, or generally having large Hamming distance for subsets of packet which are more likely to be used for reconstruction (for example: the first sent) - increasing the probability of straightforward decoding. The choice of the bit positions for packets also influences performance (they should be far from each other). Additionally, using a Pseudorandom Number Generator initialized with a cryptographic key to choose $f$ would allow to simultaneously encrypt the message - the used key would be essential to perform reconstruction.

The main limitation of discussed sequential decoding approach is relatively small number of packets to reconstruct from ($< 20$), while FC does not have such limitation. It suggests to consider some hybrid approaches: small packets reconstructed into larger packets by JRC, which are then reconstructed into the message by FC.

Finally, this paper assumes (a posteriori) knowledge of damage levels ($\epsilon$), what requires some estimation process. The redundancy of received damaged packets should allow for this purpose, suggesting that this estimation could be made as a part of decoding process: we could start with some arbitrary $\epsilon_i$ values, and try to modify them accordingly to already decoded part. Effective exploitation and analysis of above possibilities requires further work.

References

[1] I. S. Reed, G. Solomon, ”Polynomial Codes over Certain Finite Fields”, Journal of the Society for Industrial and Applied Mathematics (SIAM) 8 (2): 300-304 (1960),

[2] M. Luby, ”LT codes”, Proc. 43rd Ann. IEEE Symp. on Foundations of Computer Science, 16-19 November 2002, pp. 271-282 (2002),

[3] A. Shokrollahi, ”Raptor Codes”, Transactions on Information Theory (IEEE) 52 (6): 2551-2567 (2006),

[4] A. F. Molisch, N. B. Mehta, J. S. Yedidia, J. Zhang, ”Performance of fountain codes in collaborative relay networks”, Wireless Communications, IEEE Transactions on, 6(11), 4108-4119 (2007),

[5] Y. Lin, B. Liang, B. Li, ”Data persistence in large-scale sensor networks with decentralized fountain codes”, INFOCOM 2007, 26th IEEE International Conference on Computer Communications (2007),
REFERENCES

[6] H. Kushwaha, Y. Xing, R. Chandramouli, H. Heffes, "Reliable multimedia transmission over cognitive radio networks using fountain codes", Proceedings of the IEEE, 96(1), 155-165 (2008),

[7] A. G. Dimakis, V. Prabhakaran, K. Ramchandran, "Distributed fountain codes for networked storage", Acoustics, Speech and Signal Processing, IEEE International Conference on (Vol. 5, pp. V-V). IEEE (2006),

[8] P. Korus, J. Bialas, A. Dziech. "A new approach to high-capacity annotation watermarking based on digital fountain codes." Multimedia Tools and Applications 68.1: 59-77 (2014),

[9] I. M. Jacobs, E. R. Berlekamp, "A lower bound on to the distribution of computation for sequential decoding", IEEE Transactions on Information Theory, IT-13:167-174 (1967),

[10] A. Renyi, "On measures of information and entropy", Proceedings of the fourth Berkeley Symposium on Mathematics, Statistics and Probability 1960. pp. 547-561 (1961),

[11] J. Duda, P. Korus, "Correction Trees as an Alternative to Turbo Codes and Low Density Parity Check Codes", arXiv: 1204.5317 (2012),

[12] J. Duda, https://github.com/JarekDuda/JointReconstructionCodes ,

[13] D.J.C. MacKay, "Fountain codes." IEE Proceedings-Communications 152.6: 1062-1068 (2005),

[14] J. Duda, https://github.com/JarekDuda/DeletionChannelPracticalCorrection ,

[15] R. G. Gallager, Low Density Parity Check Codes (PDF), Monograph, M.I.T. Press (1963).