On-Line Fountain Codes with Low Overhead

Yuval Cassuto, *Senior Member, IEEE,* and Amin Shokrollahi, *Fellow, IEEE*

***Abstract*— An on-line fountain code is defined as a fountain code for which an optimal encoding strategy can be found efficiently given any instantaneous decoding state. This property is important for data distribution in practical networks. In this paper we formalize the problem of on-line fountain code construction, and propose new on-line fountain codes that outperform known ones in having factor 3-5 lower redundancy overhead. The bounding of the code overhead is carried out using analysis of the dynamics of random-graph processes.**

***Index Terms*— Fountain codes, rateless codes, on-line codes, codes with feedback, random graphs.**

I. Introduction

Fountain coding was proposed [3] for efficient distribution of data in lossy networks, with the goal to allow information transmission that is oblivious to losses of individual packets. Fountain codes that meet this goal with negligible overhead were found [10], and later improved in complexity [11]. These works, and others that followed, specify methods to encode (and decode) data blocks at the sender (and receiver) nodes, and prove upper bounds on the average overhead in the case of random losses.

Low overhead is clearly an important code-design objective, but some applications may find it insufficient on its own, with system performance being dominated by other properties of the code. The long code blocks and fixed pre-defined encoding procedures of the aforementioned fountain codes result in high decoding latency, and no way for the receivers to control or even monitor the progress of the data reception. In addition, packet losses in the network may not be all random, further exacerbating the risks of a long batch transmission designed for pure-random losses. A practical fountain code should thus balance low redundancy overhead with an *on-line* ability to adapt the code to instantaneous network conditions.

In the framework developed in this paper, a fountain code is called *on-line* if given an arbitrary decoding state at the receiver, the optimal coding strategy for the encoder can be found efficiently. This is a much stronger property than conventional fountain codes, which only guarantee optimality for the initial state of decoding. The importance of the on-line property is that it guarantees optimal performance even at states that differ significantly from the expected under random losses, e.g. due to an adversary or extremely unfortunate circumstances. As it turns out, there are known fountain codes with the on-line property such as *growth codes* [8] and *real-time oblivious codes* [2], (see also [5]). However, these codes suffer from very high redundancy overheads.

To make the on-line property usable in a communication setup, one needs to feed decoding-state information from the receiver back to the sender. That way the sender is able to adapt the encoding strategy from the current one used to the optimal one calculated from the decoding state. A feedback transmission is needed only when the decoding state changes so much that the optimal encoding strategy needs to change at the sender. The frequency at which this happens is presented as part of the code evaluation in Section VI. In particular, our on-line fountain codes use less feedback than the previously known on-line fountain schemes [8], [2]. We note that in practical scenarios feedback can be limited even further (with occasional slight deviations from optimality) without significant loss of performance. This tradeoff between feedback-use and performance has been a fertile ground for prior research on schemes that successfully reduce overhead using feedback [9], [7], [12]. Another important consideration for the proposed codes is how to operate them in a point-to-multipoint (multicast) setup, where the states of multiple receivers need to be accommodated into the encoder strategy. While we do not consider this extension in the present paper, we believe that multicasts can similarly benefit from on-line fountain codes under some mild assumptions on the loss rates. This belief comes mainly from the fact that our codes only use *uniform symbol distributions* at the encoder, thus making the joint optimization for multiple receivers possible and tractable.

The main idea behind the new codes is a new representation of the decoding state that is more amenable to algorithmic reasoning on the current state than the standard representation as a bi-partite graph. This pays in the ability to efficiently compute the optimal strategy, but costs in increased overhead due to discarding of code symbols that cannot be represented at the time of their arrival. The resulting codes attain the on-line property with significantly lower overhead, between factor 3 and factor 5 lower compared to the previously known codes [8],[2]. This significant improvement is achieved by representing the decoding state as a *uni-partite* graph, and using the graph structure to efficiently find the optimal coding strategy at the current state. The simple graph representation also allows to analyze the coding overhead, building on fundamental results from random-graph theory [6]. The contributions of the paper include the first formalization of the on-line property for fountain codes (Section II), the development of the uni-partite view of fountain codes (Section III), an on-line fountain code construction using an efficient algorithm to find the optimal coding strategy (Section IV), and bounding the overhead of a simplified construction by analyzing the dynamics of random graphs (Section V). Finally, the performance of the new codes in communication setups is evaluated through simulation (Section VI).

Yuval Cassuto is with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa Israel (email: ycassuto@ee.technion.ac.il). Part of the research was done while at the School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL).

Amin Shokrollahi is with the School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), Station 14, CH-1015 Lausanne, Switzerland (e-mail: amin.shokrollahi@epfl.ch).

Part of this work was presented at the 2011 Information Theory Workshop, Paraty Brazil [4].

This work was supported in part by an EU CIG grant, by an Israel-India ISF-UGC project, and by the Israel Ministry of Science and Technology.

There are many potential applications for on-line fountain codes. One is data distribution in the presence of intermittent adversaries, which bring receivers to arbitrarily bad decoding states and then leave them to recover. Another important application is for distributed storage, where code symbols are distributed among multiple nodes, and the on-line property gives sufficient transparency to control the long-term recoverability of the data given an instantaneous node-failure state.

II. On-Line Fountain Codes

Block rateless fountain codes, such as LT [10] and Raptor [11] codes, are designed for *batch* transmission of coded symbols. These codes aim at minimizing the number of coded symbols required to successfully decode the entire code block with high probability. Given a current state of code symbols already received at a receiver node, batch fountain codes do not address the minimization of code-symbol transmissions until complete decoding. It has been recognized in prior work that not considering the current state of already received symbols results in sub-optimal performance in many practical scenarios. Several works have proposed adaptations to batch fountain codes, mostly LT codes, that do consider the current decoding state for the choice of subsequent code symbols to transmit. A partial list of such coding schemes includes [9], [7], [12]. The outcome of these schemes is that faster decoding can be achieved by feeding decoding-state information from the receiver back to the transmitter. In this paper we make another step in this direction and look for coding schemes for which the *provably optimal* coding strategy can be found given an arbitrary decoding state. The ability to efficiently find the optimal coding strategy given the current state is very important in practical applications, since it guarantees fast decoding at any point of time without any assumption on the past behavior of the encoder and channel. Fountain codes with known optimal coding strategies given the current decoding state are called herein *on-line* fountain codes. No matter how unusual or unfortunate previous encodings and transmissions have been, from the current state onward on-line fountain codes guarantee optimal performance. This property is maintained even if the code-symbol losses till this point are maliciously set by an adversary. Before giving a formal definition of the on-line property, we make the terms *fountain code* and *coding strategy* more precise.

Definition 1. *A **fountain code** has a block of k input symbols, an encoder and a decoder. The encoder generates code symbols by taking eXclusive-OR (XOR) operations on subsets of input symbols. The decoder processes the received code symbols and outputs the k input symbols.*

Definition 2. *A **coding strategy** of a fountain code is a set of specifications according to which the encoder chooses the subsets of input symbols to XOR for generation of code symbols.*

A coding strategy is selected from a pre-defined universe of permissible strategies. For example, in fountain coding schemes it is common to only consider *uniform* coding strategies, which allow to vary the degree (number of operands) of the XOR operations, but require to select the XOR operands uniformly from the k input symbols. While in general non-uniform fountain codes are useful and well studied – mainly in the context of unequal error protection (UEP) codes – in this paper we restrict ourselves to uniform codes only. The reasoning behind only considering uniform fountain codes is that non-uniform codes require encoders that are much more complex in optimizing their outputs, and much more informed about the decoding states of the receivers.

Definition 3. *A fountain code is called **on-line** if given an arbitrary decoding state at the receiver, the optimal coding strategy for the encoder can be found efficiently.*

It is important to clarify that *optimal* in Definition 3 refers to optimality with respect to the decoder chosen for the fountain code. That is, for a fountain code to be qualified as an on-line fountain code, one has to first specify the decoder, then there must exist an algorithm that uses the current decoding state to efficiently determine the encoding function that maximizes some measure of decoding progress. We also note that Definition 3 does not specify the mechanism by which the coding strategy is adapted in a communication setup. In particular, both of the following options are possible

1) The receiver node feeds its decoding state back to the sender. The sender node computes the optimal strategy from the state input.
2) The receiver node itself computes the optimal strategy from its decoding state. The optimal strategy is fed back to the sender node.

The decision how to perform coding-strategy updates depends upon practical circumstances, mainly considering the cost of feedback communications vs. the receiver's computation capabilities.

A fountain code with the on-line property was proposed by Kamra et. al in [8] (called growth code), and independently by Beimel et. al in [2] (called real-time oblivious code). In this code, with each received code symbol the decoder attempts to decode an input symbol not previously known to it, and discards the code symbol if this attempt fails. For this decoder, the optimal degree for a code symbol drawn by the encoder is the one that maximizes the probability that a new input symbol will become known following the receipt of the code symbol. This optimal degree was given in [8] and [2] as a function of the current number of decoded symbols at the receiver. Knowing the optimal degree immediately defines the optimal coding strategy at the sender as using this optimal degree in subsequent generations of code symbols. Since this optimal strategy is known for any decoding state (number of decoded symbols), this fountain code has the on-line property. However, in this code the on-line property comes with a prohibitive cost in redundancy. In [2] a redundancy of 100% is suggested, i.e., $2k$ received code symbols are needed to decode the k input symbols with high probability. It is possible to prove that the redundancy of this code is lower bounded by $\ln 2 \approx 0.69$. This is shown in Section V-C where we compare this known code to an improved code that achieves the on-line property with a much more reasonable cost of redundancy.

III. Fountain Codes on Uni-partite Graphs

To obtain on-line fountain codes with lower redundancy than best known codes, we now define a simple graph structure on which fountain codes can be made on-line with lower redundancy. The canonical representation of fountain codes throughout the literature is as bi-partite graphs. Nodes of one type, *input nodes*, represent the k input information symbols; nodes of another type, *code nodes*, represent the code symbols. An edge between an input node x_i and a code node y_j marks that input symbol x_i is one of the summands[1] of code symbol y_j. For example, Figure 1 depicts the following code symbols

$$y_1{=}x_1{+}x_2,\ y_2{=}x_2{+}x_3,\ y_3{=}x_1{+}x_3,\ y_4{=}x_4{+}x_5,\ y_5{=}x_6.$$

When all of the degrees of code nodes are 2 or less, the code

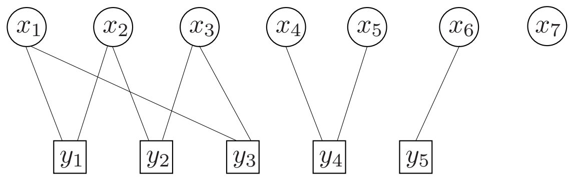

Figure 1. A fountain code as a bi-partite graph. Code symbols (square nodes) are XORs of input symbols (circle nodes).

can be equivalently represented by a uni-partite graph. For the example above, the corresponding uni-partite graph is given in Figure 2. In the uni-partite graph, two nodes are connected

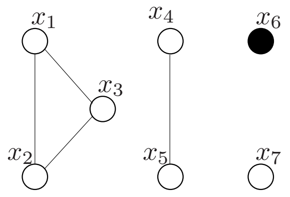

Figure 2. A fountain code as a uni-partite graph.

with an edge if there is a code symbol that is the sum of the corresponding two input symbols. A node is colored black if it is known to the decoder (and white otherwise). There are no edges incident on black nodes. In the example, symbol x_6 is black thanks to the code symbol $y_5 = x_6$.

Definition 4. *A (uni-partite)* ***decoding graph*** *is a representation of a collection of code symbols comprising*

1) *A code symbol x_i for every black node i.*
2) *A code symbol $x_i + x_j$ for every edge (i, j).*

The advantage of the uni-partite view of a fountain code is that it can carry the full state of code symbols with degree up to 2, over a simple structure that allows finding the optimal coding strategy using graph-theoretic notions. Note that the fact that the decoding graph at the *receiver* only represents degree-2 code symbols does not mean that the the *sender* is restricted to using exclusively degree-2 code symbols.

[1]For convenience we regard the XOR operations as additions modulo 2.

A. Implicit edges and the the connected-component enumerator

In addition to edges representing explicit code symbols received from the channel, edges can be added between every pair of nodes that is *connected* in the graph. These edges can be derived by summing all code symbols along a connecting path, canceling out all intermediate input symbols on the path. For example, the explicit edge between x_1 and x_3 in Figure 2 is redundant, since it could have been obtained by summing up the symbols $(x_1 + x_2) + (x_2 + x_3)$. Thus from the perspective of the receiver, the state of decoding is fully described by the *connected components* in the decoding graph and the set of black nodes. (A connected component in a graph is a set of vertices, all of which are connected by paths of edges, and none of which is connected to any vertex outside the component. The number of vertices in this set is called the *size* of the connected component.). The remainder of the section is devoted to setting up notation for characterizing the decoding state through the decoding graph's connected components. This notation will be used in the next sections to define and analyze coding strategies for on-line fountain codes.

Suppose the decoding graph at a given instant has A black nodes and A_i connected components of size i, where i ranges from 1 to the maximal component size ($\leqslant k$). Then the decoding state can be represented by the *component enumerator polynomial*, given by

$$\mathcal{A}(x) = A + \sum_{i=1}^{k} A_i x^i.$$

The total number of components is

$$\#\text{components} = \sum_{i=1}^{k} A_i = \mathcal{A}(1) - A. \qquad (1)$$

Since the connected components partition the graph nodes that are not black,

$$\mathcal{A}'(1) = \sum_{i=1}^{k} i A_i = k - A. \qquad (2)$$

The average component size in the graph can thus be calculated as

$$L = \frac{k - A}{\#\text{components}} = \frac{k - A}{\mathcal{A}(1) - A} = \frac{\mathcal{A}'(1)}{\mathcal{A}(1) - \mathcal{A}(0)}.$$

There is a clear correspondence between the structure of the decoding graph and the dimension of the linear subspace spanned by the received code symbols.

Proposition 1. *Let $G \subset \{0, 1\}^k$ be the linear subspace spanned by the code symbols of a decoding graph $\mathcal{G}$. The dimension of G is given by*

$$\dim(G) = k - \#\text{components}(\mathcal{G}).$$

Proof: Each black node in $\mathcal{G}$ contributes one independent vector to the span of the code symbols. In addition, each connected component of size i contributes to the span $i - 1$ independent vectors. Any $i-1$ edges of a spanning tree for the component can give the $i-1$ independent degree-2 vectors. Summing the contributions of black and white nodes, we get

$$\begin{aligned}\dim(G) &= A + \sum_{i=1}^{k}(i-1)A_i \\ &= A + \sum_{i=1}^{k} iA_i - \sum_{i=1}^{k} A_i = k - \#\text{components},\end{aligned}$$

where the last equality follows from the second equality of (2) and the first equality of (1). ■

Proposition 1 implies that increasing the dimension of the code symbols' span at the decoder is equivalent to eliminating a connected component from the decoding graph. This feature will be the key for finding the optimal coding strategy in the on-line fountain code presented in the next section.

After defining the component-enumerator polynomial and detailing some of its properties, we turn to endow it with an operator that will become useful later.

Definition 5. *Given a component-enumerator polynomial $\mathcal{A}(x)$, let $\mathcal{A}^{\downarrow t}(x)$ be the polynomial with coefficients specified as*

$$A_i^{\downarrow t} = \begin{cases} \max(A_t - 1, 0) & i = t \\ A_i & \text{otherwise} \end{cases}$$

thus $\mathcal{A}^{\downarrow t}(x)$ has all of its coefficients identical to $\mathcal{A}(x)$, except for the coefficient A_t which is decreased by one, if not already zero.

IV. An On-Line Fountain Code Construction

The on-line fountain code proposed in this section follows from the simple observation that if a code symbol corresponds to an edge added between a black node and a white node in the decoding graph, then all the nodes in the white node's connected component will be colored in black. For example, in Figure 2 a symbol x_4+x_6 would decode x_4 and x_5, both in the component of x_4. First x_4 is recovered by subtracting the known x_6 from x_4+x_6, and then x_5 is found by subtracting x_4 from x_4+x_5. Another important simple observation is that if a code symbol corresponds to an edge added between two white nodes, then the connected components of the two nodes (if not already in the same component) are merged to a single component.

In our on-line fountain code, the coding strategy will seek to maximize the probability of either coloring a component in black, or merging two distinct components. Both options eliminate a connected component in the graph, and by Proposition 1 either will increase by one the dimension spanned by the code symbols. We refer to such increase in dimension as *decoding progress*. Hence given a current decoding state presented as a decoding graph $\mathcal{G}$, decoding progress is achieved if a newly received code symbol results in either of the following two outcomes

1) An edge coloring a component in black.
2) An edge connecting two distinct components.

All other cases either provide redundant information (symbol sums already known to the receiver), or cannot be handled by the uni-partite scheme (sum relations involving more than two components). We thus seek to maximize the probability that one of the events 1 and 2 occurs, given the instantaneous decoding state. A more concrete (but equivalent) definition of events 1 and 2, respectively, is

Case 1 A received code symbol sums an odd number of input symbols from *a single* component, with other components contributing even numbers of input symbols, in addition to any number (even or odd) of black symbols.

Case 2 A received code symbol sums odd numbers of input symbols from *two* components, with other components contributing even numbers of input symbols, in addition to any number (even or odd) of black symbols.

Case 1 and Case 2 result in 1 and 2 above, respectively, because the sum of an even number of summands from the same component is known at the decoder, and can be canceled; also known is the sum of any number (even or odd) of black symbols.

Denote by $\mathsf{P}_1(m,\mathcal{A})$ and $\mathsf{P}_2(m,\mathcal{A})$ the probabilities that a uniformly selected degree-m code symbol results in a Case 1 and Case 2 event, respectively. The argument $\mathcal{A}$ is the component enumerator polynomial of the graph $\mathcal{G}$ at the receiver. Then we can define our on-line fountain code as follows.

Construction 1. *The code is defined on a block of k input symbols.*

Decoder:

1) *Initialize the decoding graph as a graph with k white nodes and no edges.*
2) *If a received code symbol falls under Case 1 or Case 2, update the decoding graph. Otherwise discard the code symbol.*

Encoder:

Given the current decoding state presented as a graph with component enumerator polynomial $\mathcal{A}$, the coding strategy is set to drawing uniformly at random code symbols with degree $\hat{m}$, where

$$\hat{m} = \text{argmax}_m \left[\mathsf{P}_1(m,\mathcal{A}) + \mathsf{P}_2(m,\mathcal{A})\right]. \tag{3}$$

Note first the low complexity required to decode Construction 1. An incoming code symbol of any degree is decoded by "peeling" from it black symbols and XORs of pairs from a connected component. If the pair-XORs are stored minimally as balanced spanning trees of connected components, then each peeling of a pair requires $\log$ the component size XOR operations. To show that Construction 1 defines an on-line fountain code, we need to prove that the coding strategy given in (3) is optimal.

Proposition 2. *Let $G \subset \{0,1\}^k$ be the linear subspace spanned by the code symbols of the current decoding graph $\mathcal{G}$ with component enumerator $\mathcal{A}$. Let $\boldsymbol{v} \in \{0,1\}^k$ be the next code symbol drawn by the encoder, and $G' \subset \{0,1\}^k$ the linear subspace spanned by the code symbols after $\boldsymbol{v}$ is processed by the decoder. Then the expected dimension of G' is maximized if $\boldsymbol{v}$ is drawn according to the coding strategy of* (3).

Proof: If $\boldsymbol{v}$ falls under Case 1 or Case 2, then exactly one connected component is eliminated from the decoding graph. In Case 1 a component turns black, and in Case 2 two components are merged to one. According to Proposition 1, eliminating a component amounts to increasing the dimension by one. If neither of Case 1 or Case 2 applies to $\boldsymbol{v}$, then there is no change in the connected components, and the dimension of G' remains the same as the dimension of G. So overall, for degree m code symbols we have

$$\begin{aligned} E\left[\dim(G')\right] = \dim(G) + 1 \cdot [\mathsf{P}_1(m,\mathcal{A}) + \mathsf{P}_2(m,\mathcal{A})] \\ + 0 \cdot [1 - \mathsf{P}_1(m,\mathcal{A}) - \mathsf{P}_2(m,\mathcal{A})]. \end{aligned}$$

And clearly maximizing the expectation is achieved by maximizing $\mathsf{P}_1(m,\mathcal{A}) + \mathsf{P}_2(m,\mathcal{A})$ over all m. ■
To calculate the decoding-progress probabilities $\mathsf{P}_1(m,\mathcal{A})$ and $\mathsf{P}_2(m,\mathcal{A})$, we need to count the number of degree-m code symbols that result in Case 1 and Case 2 symbols, respectively.

Given a component enumerator polynomial $\mathcal{A}(x)$, let $N_1(m,\mathcal{A})$ denote the number of degree-m code symbols that color a component in black (Case 1), and let $N_2(m,\mathcal{A})$ similarly denote the number of degree-m code symbols that connect two distinct components (Case 2). Assuming the degree-m code symbol is drawn uniformly at random, we have the relations

$$\mathsf{P}_l(m,\mathcal{A}) = \frac{N_l(m,\mathcal{A})}{\binom{k}{m}}, \text{ for } l \in \{1,2\}. \tag{4}$$

A. Exact calculation of P_1

We start off with counting Case 1 combinations. We establish a recursive formula for this count in a theorem following a series of lemmas. The following definitions will turn out useful.

Definition 6. *Denote by $\mathsf{OEs}(j,\mathcal{A})$ the number of combinations of j symbols that have an odd number of symbols in one component, and even numbers (including zero) of symbols in all other components, given a component enumerator $\mathcal{A}(x)$.*

The symbol OEs chosen for Definition 6 stands for *Odd-Evens*.

Definition 7. *Denote by $\mathsf{Es}(j,\mathcal{A})$ the number of combinations of j symbols that have even numbers (including zero) of symbols in all components, given a component enumerator $\mathcal{A}(x)$.*

The symbol Es chosen for Definition 7 stands for *Evens*. The first lemma ties the function OEs from Definition 6 with Es from Definition 7.

Lemma 3. *Let $\mathcal{A}(x)$ be a component enumerator polynomial, and j an integer satisfying $j \leqslant k$. Then*

$$\mathsf{OEs}(j,\mathcal{A}) = \sum_{\substack{i=1 \\ i \text{ odd}}}^{j} \sum_{t=i}^{k} A_t \binom{t}{i} \mathsf{Es}(j-i, \mathcal{A}^{\downarrow t}). \tag{5}$$

Proof: i in the outer sum is the number of input symbols in the component that contains an odd number of input symbols in the degree-j code symbol; t in the inner sum is the size of this component. The function Es is invoked on the remaining $j-i$ input symbols and on the polynomial $\mathcal{A}_{\downarrow t}$, to exclude the component chosen to have an odd number of symbols. The multiplicative coefficient of Es is the number of size-t components in the graph, times the number of i-subsets of the size-t component. ■
The next lemma gives a recursive formula for Es. Counting Es is established by successively choosing a connected component with an even non-zero number of symbols, then removing this component from the enumerator and continuing recursively.

Lemma 4. *Let $\mathcal{A}(x)$ be a component enumerator polynomial, and j an integer satisfying $j \leqslant k$. Define a 3-argument function $\mathsf{Es}(j,\mathcal{A},d)$ with the following relation to the original function $\mathsf{Es}(j,\mathcal{A})$*

$$\mathsf{Es}(j,\mathcal{A},1) \triangleq \mathsf{Es}(j,\mathcal{A}).$$

Then $\mathsf{Es}(j,\mathcal{A})$ can be calculated by invoking the following recursive formula with $d=1$:
for $j>0$

$$\mathsf{Es}(j,\mathcal{A},d) = \frac{1}{d} \sum_{\substack{i=2 \\ i \text{ even}}}^{j} \sum_{t=i}^{k} A_t \binom{t}{i} \mathsf{Es}(j-i, \mathcal{A}^{\downarrow t}, d+1). \tag{6}$$

for $j=0$

$$\mathsf{Es}(0,\mathcal{A},d) = 1 \qquad .$$

Proof: Let us first naïvely apply a recursion similar to (5) toward the count of $\mathsf{Es}(j,\mathcal{A})$. Since correction will be needed, we use $\tilde{\mathsf{Es}}(j,\mathcal{A})$ in the formula.

$$\tilde{\mathsf{Es}}(j,\mathcal{A}) = \sum_{\substack{i=2 \\ i \text{ even}}}^{j} \sum_{t=i}^{k} A_t \binom{t}{i} \tilde{\mathsf{Es}}(j-i, \mathcal{A}^{\downarrow t}). \tag{7}$$

The result of the recursion in (7) is that each combination of j symbols falling into r components, with non-zero even numbers of symbols in each, is counted $r!$ times. This is because an r-component combination is multiply-counted in every order of successive selection among the r components. So to get a correct count, we need to divide by $r!$ the number of combinations with r components, for each possible r.
This correction is achieved by the $1/d$ factor adjoined to (6), such that an r-component combination count is successively normalized by $\frac{1}{1} \cdot \frac{1}{2} \cdot \ldots \cdot \frac{1}{r} = \frac{1}{r!}$. ■
After showing exact recursive counts of $\mathsf{OEs}(j,\mathcal{A})$ and $\mathsf{Es}(j,\mathcal{A})$ in Lemmas 3 and 4, respectively, we turn to show how to use these counts to calculate the progress probability $\mathsf{P}_1(m,\mathcal{A})$.

Theorem 5. *Let $\mathcal{A}(x)$ be a component enumerator polynomial, and m an integer satisfying $m \leqslant k$. Then the number of Case 1 degree-m code symbols can be counted exactly, and is given by*

$$N_1(m,\mathcal{A}) = \sum_{s=0}^{m-1} \binom{A}{s} \mathsf{OEs}(m-s,\mathcal{A}).$$

Proof: By definition of Case 1, any number s of black symbols can be included in the code symbol, and the requirement on the remaining symbols is exactly as stated in the definition of OEs (Definition 6). Hence we sum over all s, take all possible size-s subsets as black symbols, and apply OEs to the remaining $m-s$ symbols. ■

Now that we established the count of $N_1(m,\mathcal{A})$, we can obtain $\mathsf{P}_1(m,\mathcal{A})$ using (4).

B. Exact calculation of P_2

Moving to count Case 2 combinations, we define another useful combinatorial function.

Definition 8. *Denote by* $\mathsf{OOEs}(j,\mathcal{A})$ *the number of combinations of* j *symbols that have odd numbers of symbols in* <u>*two*</u> *components, and even numbers (including zero) of symbols in all other components, given a component enumerator* $\mathcal{A}(x)$*.*

The symbol OOEs stands for *Odd-Odd-Evens*. A recursive count for $\mathsf{OOEs}(j,\mathcal{A})$ is given in the next lemma.

Lemma 6. *Let* $\mathcal{A}(x)$ *be a component enumerator polynomial, and* j *an integer satisfying* $j \leqslant k$*. Then*

$$\mathsf{OOEs}(j,\mathcal{A}) = \frac{1}{2}\sum_{\substack{i=1\\ i\text{ odd}}}^{j}\sum_{t=i}^{k} A_t\binom{t}{i}\mathsf{OEs}(j-i,\mathcal{A}^{\downarrow t}). \quad (8)$$

Proof: i in the outer sum is the number of input symbols in the *first* (out of the two) component that contains an odd number of input symbols in the degree-j code symbol; t in the inner sum is the size of this component. After choosing one component with odd number of symbols, the remaining symbols need to divide to an odd number in one component, and even numbers in all the rest. Hence the function $\mathsf{OEs}(j,\mathcal{A})$ from (5) can be used. The $1/2$ factor cancels the double count of each combination: there are two components with odd numbers of symbols, and identical combinations are obtained by reversing the selection order of these components. ■

Now we reach a theorem similar to Theorem 5, this time for $N_2(m,\mathcal{A})$.

Theorem 7. *Let* $\mathcal{A}(x)$ *be a component enumerator polynomial, and* m *an integer satisfying* $m \leqslant k$*. Then the number of Case 2 degree-*m *code symbols can be counted exactly, and is given by*

$$N_2(m,\mathcal{A}) = \sum_{s=0}^{m-1}\binom{A}{s}\mathsf{OOEs}(m-s,\mathcal{A}).$$

Proof: The proof is essentially the same as in Theorem 5. s, as before, is the number of black symbols in the degree-m code symbol. After choosing the s black symbols, the requirement on the remaining symbols is exactly as stated in the definition of OOEs (Definition 8). ■

Now that we established the count of $N_2(m,\mathcal{A})$, we can obtain $\mathsf{P}_2(m,\mathcal{A})$ using (4).

The outcome of the exact calculations of $\mathsf{P}_1(m,\mathcal{A})$ and $\mathsf{P}_2(m,\mathcal{A})$ through Theorems 5 and 7 is that we can efficiently implement the optimal on-line encoder specified in (3). Each recursive call to a combinatorial function is invoked with a distinct pair of arguments $j,\mathcal{U}$. In addition, the number of different enumerators $\mathcal{U}$ used throughout the recursion is at most the number of components, because of the relation $\mathcal{A}^{\downarrow t}$ in each chain. Therefore, the complexity of calculating $N_l(m,\mathcal{A})$ is at most m times the number of components in $\mathcal{A}$. This implies low complexity in practice because initially when the number of components is large the optimal m is typically small, and the optimal m is typically large only later in decoding when the number of components is already small. One issue to consider here is the need to evaluate large binomial coefficients, which may impose numerical difficulties. This can be solved in practice by replacing the exact binomial coefficients with well known approximations that are more numerically stable.

C. Example of optimal on-line encoding

The following (small) example serves to demonstrate how degrees are optimally chosen given the current decoding state.

Example 1. *Suppose the current decoding state of a certain receiver is given by the component enumerator polynomial* $\mathcal{A}(x) = 2 + x^2 + x^4$*. In other words, out of the* $k = 8$ *input symbols,* 2 *are already decoded (black), and the remaining* 6 *are divided to one component of size* 2 *and one of size* 4*. For each possible degree* $m \in \{1,\ldots,k\}$*, the receiver computes the probabilities* $\mathsf{P}_1(m,\mathcal{A})$*,* $\mathsf{P}_2(m,\mathcal{A})$*, and the sum thereof. These values are given in Table* I*. The outcome of these calculations*

m	1	2	3	4	5	6	7	8
P_1	0.75	0.428	0.464	0.571	0.464	0.428	0.75	0
P_2	0	0.286	0.286	0.229	0.286	0.286	0	0
P_1+P_2	0.75	0.714	0.75	**0.8**	0.75	0.714	0.75	0

TABLE I
DECODING-PROGRESS PROBABILITIES FOR DIFFERENT DEGREES m

is that the optimal degree for the current state is $\hat{m} = 4$*, which gives* $\mathsf{P}_1 + \mathsf{P}_2 = 0.8$ *(bold in Table* I*). To see how this value is obtained, we count* $N_1(4,\mathcal{A})$ *and* $N_2(4,\mathcal{A})$*. To get an odd number of symbols in one component (Case 1) for* $m = 4$*, we have the following possibilities:* 1 *black and* 3 *in the size-*4 *component,* 1 *black,* 2 *in the size-*2 *component and* 1 *in the size-*4 *component, or* 1 *black,* 2 *in the size-*4 *component and* 1 *in the size-*2 *component. These amount to* $2\cdot 4 = 8$, $2\cdot 1\cdot 4 = 8$ *and* $2\cdot 6\cdot 2 = 24$*, respectively. Hence* $N_1(4,\mathcal{A}) = 40$*. To get an odd number of symbols in two components (Case 2), we have the following possibilities:* 3 *in the size-*4 *component and* 1 *in the size-*2 *component, or* 2 *blacks,* 1 *in the size-*2 *component and* 1 *in the size-*4 *component. These amount to* $4\cdot 2 = 8$*, and* $1\cdot 2\cdot 4 = 8$*, respectively. Hence* $N_2(4,\mathcal{A}) = 16$*. Since* $\binom{k}{4} = 70$*, we get*

$$\mathsf{P}_1(4,\mathcal{A}){+}\mathsf{P}_2(4,\mathcal{A}) = \frac{N_1(4,\mathcal{A}){+}N_2(4,\mathcal{A})}{\binom{k}{4}} = \frac{40{+}16}{70} = 0.8.$$

V. A SIMPLIFIED ON-LINE FOUNTAIN CODE

In the previous section, an on-line fountain code was given where the receiving clients can efficiently calculate the optimal degree at any stage of decoding. In this section, we leave the regime of strictly optimal degree choices, and consider a simplified on-line fountain code that is simpler both for implementation and for analysis. With the simplified code, it is shown that a simpler criterion than maximizing $\mathsf{P}_1(m,\mathcal{A})+$

$\mathsf{P}_2(m, \mathcal{A})$ precisely can with high probability give the optimal encoding degrees at intermediate decoding stages. The main result pertaining to the simplified code is an upper bound on the redundancy overhead, which is shown to be much lower than known on-line fountain codes.

A. Specification

In the simplified scheme, the encoding is first coarsely divided to two phases called *build-up* and *completion*. In the build-up phase the degree is set to constant 2, and in the completion phase the client uses a simplified instantaneous-degree optimization, which only depends upon the number of decoded (black) symbols. Note that while the chosen degrees depend on the instantaneous number of decoded symbols, these degrees do *not* optimize solely for higher probability of symbol decoding, but rather jointly optimize the symbol-decoding probability and the connectivity of the decoding graph. This is in contrast with known schemes [8], [2] that only optimize for the former (and are thus much more costly in terms of overhead). A description of the two-phase coding procedure now follows.

1) Build-up

At the build-up phase, the sender transmits uniform code symbols of degree 2. These symbols add edges to the decoding graph at the receiver. The build-up phase continues until a connected component of size $|D| = \beta_0 k$ exists in the graph ($0 < \beta_0 < 1$ is a parameter). Then the sender colors the large component in black by sending uniform degree-1 symbols until hitting the largest component for the first time. The expected number of degree-1 symbols required before hitting a component of size linear in k is a small constant.

2) Completion

Given a graph with βk black vertices, choose the degree $\hat{m}$ that maximizes the sum probability of cases 1' and 2' below.

Case 1' A received symbol sums a single white symbol with $m-1$ black symbols.

Case 2' A received symbol sums two white symbols with $m-2$ black symbols.

Symbols that fall into cases 1' and 2' are used to update the decoding graph. Other symbols are discarded.

A few remarks on the completion phase are now in place. Case 1' decodes at least one white symbol (the white symbol's component), and Case 2' adds an innovating edge, i.e. an edge that increases the dimension of the received subspace, unless the two white symbols are in the same component. The motivation to consider cases 1' and 2', and not the more elaborate cases 1 and 2 of Section IV is the following. The appeal of cases 1' and 2' is that their probabilities can be calculated based on β alone (as detailed in the next paragraph), without need to know the complete component enumerator. If the white connected components are fairly small compared to k, then cases 1' and 2' are good approximations of cases 1 and 2 of Section IV, due to the low probability of having multiple symbols in the same component. This is shown analytically in Section V-B.

The two cases 1' and 2' at the completion phase are illustrated in Figure 3. Note that edges between white nodes do exist in the graph, but the classification to Case 1' and 2' ignores these edges.

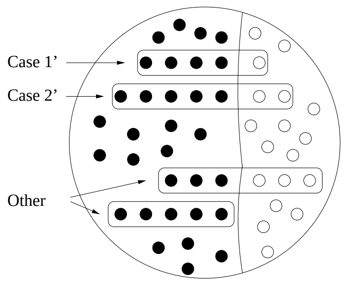

Figure 3. Case 1' and Case 2' sought in the completion phase. Other cases (showed at the bottom) result in discarding the received symbol.

A formal specification of the simplified code is now given as Construction 2.

Construction 2. *The code is defined on a block of k input symbols.*

Decoder:

1) *Initialize the decoding graph as a graph with k white nodes and no edges.*
2) *If a received code symbol falls under Case 1' or Case 2', update the decoding graph. Otherwise discard the code symbol.*

Encoder:

In the build-up phase: send uniformly distributed degree-2 symbols.

After the build-up phase: send uniformly distributed degree-1 symbols until the largest component turns black.

In the completion phase: given the current decoding state presented as a fraction β of black symbols, the coding strategy is set to drawing uniformly at random code symbols with degree $\hat{m}$, where

$$\hat{m} = \operatorname{argmax}_m \left[\mathsf{P}_{1'}(m,\beta) + \mathsf{P}_{2'}(m,\beta)\right], \tag{9}$$

and $\mathsf{P}_{1'}(m,\beta)$ and $\mathsf{P}_{2'}(m,\beta)$ are the probabilities of Case 1' and Case 2', respectively.

Note that $\mathsf{P}_{1'}$ and $\mathsf{P}_{2'}$ used in Construction 2 depend on m and β, the fraction of black symbols, but not on the full component enumerator as in Construction 1 of Section IV. Assuming selection of m symbols from the size-k block with replacement, the expressions for $\mathsf{P}_{1'}$ and $\mathsf{P}_{2'}$ are given by

$$\mathsf{P}_{1'}(m,\beta) = \binom{m}{1}\beta^{m-1}(1-\beta), \tag{10}$$

$$\mathsf{P}_{2'}(m,\beta) = \binom{m}{2}\beta^{m-2}(1-\beta)^2. \tag{11}$$

In equations (10),(11) we see the complexity benefits of using the simplified code. Instead of evaluating the full recursive formulas for $\mathsf{P}_1, \mathsf{P}_2$ in Section IV, now we have a closed-form expression for the surrogate probabilities $\mathsf{P}_{1'}, \mathsf{P}_{2'}$. The values of $\mathsf{P}_{1'} + \mathsf{P}_{2'}$ as a function of β for different m values are plotted in Figure 4.

The following theorem provides a lower bound on $\mathsf{P}_{1'} + \mathsf{P}_{2'}$, which will be used for analysis in the next sub-section.

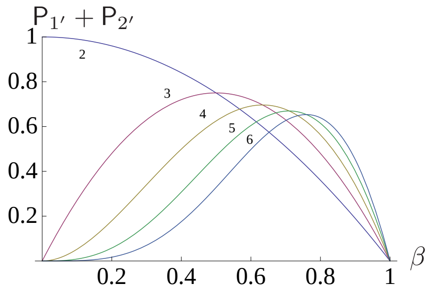

Figure 4. The sum probability of cases 1' and 2' as a function of the fraction of black symbols. Plotted for $m = 2, 3, 4, 5, 6$.

Theorem 8. *For any β, there exists a $\hat{m}$ such that*

$$\mathsf{P}_{1'}(\hat{m}, \beta) + \mathsf{P}_{2'}(\hat{m}, \beta) > (1 + \sqrt{2})e^{-\sqrt{2}} = 0.5869, \quad (12)$$

and $\hat{m}$ is the unique m that satisfies

$$\frac{\sqrt{(m-1)(m-2)}}{\sqrt{2} + \sqrt{(m-1)(m-2)}} \leqslant \beta < \frac{\sqrt{m(m-1)}}{\sqrt{2} + \sqrt{m(m-1)}}.$$

Proof: The proof will divide into two main parts. First we prove the bound when the value β is taken from an infinite discrete sequence of real values. Then we show that the first part implies the bound at all points of the real-line segment $[0, 1)$. The sequence of discrete real values we consider in the first part contains all the values β that satisfy

$$\mathsf{P}_{1'}(m, \beta) + \mathsf{P}_{2'}(m, \beta) = \mathsf{P}_{1'}(m+1, \beta) + \mathsf{P}_{2'}(m+1, \beta), \quad (13)$$

for some integer m. These β values are the transition points of the encoder from degree m to degree $m + 1$, seen in Figure 4 as the intersections of two curves with adjacent labels. When we substitute $\mathsf{P}_{1'}(m, \beta)$ from (10) and $\mathsf{P}_{2'}(m, \beta)$ from (11) into (13) and solve[2] for β, we obtain

$$\beta_m = \frac{\sqrt{m(m-1)}}{\sqrt{2} + \sqrt{m(m-1)}}. \quad (14)$$

The values β_m can be regarded as a sequence $B = \{\beta_m\}_{m=1}^{\infty}$. The evaluation of the sequence B is

$$B = \{0, 0.5, 0.634, 0.710, 0.760, 0.795, ...\},$$

and it tends to 1 as m tends to infinity. To show the lower bound on points in B, we substitute the right-hand side of (14) as β into (10) and (11), and obtain

$$\mathsf{P}_{1'}(m, \beta_m) + \mathsf{P}_{2'}(m, \beta_m) = \frac{m + \sqrt{2m(m-1)} - 1}{m - 1} \cdot \left(\frac{m^2 - m - \sqrt{2m(m-1)}}{m^2 - m - 2} \right)^m. \quad (15)$$

[2]Equation (13) has in addition a trivial solution $\beta_m = 1$, for any m, which we ignore in the remainder of the proof.

Now reorganizing (15), we obtain

$$\mathsf{P}_{1'}(m, \beta_m) + \mathsf{P}_{2'}(m, \beta_m) = \left(1 + \sqrt{\frac{2m}{m-1}}\right) \cdot \left(1 - \frac{\sqrt{2}}{\sqrt{2} + \sqrt{m(m-1)}}\right)^m > \left(1 + \sqrt{\frac{2m}{m-1}}\right) \cdot \left(1 - \frac{\sqrt{2}}{\sqrt{2} + m - 0.6}\right)^m, \quad (16)$$

where the last inequality comes from the fact that $\sqrt{m(m-1)} > m - 0.6$ for $m > 1$. It is clear that both terms in the right-hand side of (16) are positive and monotone decreasing with m, and hence their product is also monotone decreasing with m. This gives the bound

$$\mathsf{P}_{1'}(m, \beta_m) + \mathsf{P}_{2'}(m, \beta_m) > \lim_{m \to \infty} \left(1 + \sqrt{\frac{2m}{m-1}}\right) \cdot \left(1 - \frac{\sqrt{2}}{\sqrt{2} + m - 0.6}\right)^m = (1 + \sqrt{2})e^{-\sqrt{2}} = 0.5869. \quad (17)$$

Now that we proved the lower bound for β values on the transitions from m to $m + 1$, we move to the second part of the proof, which extends the bound to all β values. The idea of the second part is to show that $\mathsf{P}_{1'}(m, \beta) + \mathsf{P}_{2'}(m, \beta)$ is *concave* for $\beta \in (\beta_{m-1}, \beta_m)$. Concavity implies that for all $\beta \in (\beta_{m-1}, \beta_m)$

$$\mathsf{P}_{1'}(m, \beta) + \mathsf{P}_{2'}(m, \beta) \geqslant \min\left[\mathsf{P}_{1'}(m, \beta_{m-1}) + \mathsf{P}_{2'}(m, \beta_{m-1}), \mathsf{P}_{1'}(m, \beta_m) + \mathsf{P}_{2'}(m, \beta_m)\right].$$

The concavity proof of $\mathsf{P}_{1'}(m, \beta) + \mathsf{P}_{2'}(m, \beta)$ in (β_{m-1}, β_m) is direct, using elementary calculus. We take the second derivative of $\mathsf{P}_{1'}(m, \beta) + \mathsf{P}_{2'}(m, \beta)$ with respect to β, and find that it is negative in the interval

$$1 - \frac{m + \sqrt{3m^2 - 10m + 16} - 4}{m(m-3)} < \beta < 1 - \frac{m - \sqrt{3m^2 - 10m + 16} - 4}{m(m-3)}.$$

The concavity in the desired domain is established by showing that for all positive integers $m > 3$

$$1 - \frac{m + \sqrt{3m^2 - 10m + 16} - 4}{m(m-3)} < \beta_{m-1}$$

and

$$\beta_m < 1 - \frac{m - \sqrt{3m^2 - 10m + 16} - 4}{m(m-3)}.$$

For $m = 2, 3$ similar concavity can be proved by first substituting m in (10) and (11), and then taking the second derivative and verifying its negativity in (β_{m-1}, β_m). ■

For the analysis upcoming in the next sub-section we need the following corollary, showing that $\mathsf{P}_{1'}(\hat{m}, \beta)$ and $\mathsf{P}_{2'}(\hat{m}, \beta)$ *individually* tend to constants for $\beta \to 1$, and not just their sum.

Corollary 9. *For the $\hat{m}$ that attains (12) in Theorem 8, both $\mathsf{P}_{1'}(\hat{m}, \beta)$ and $\mathsf{P}_{2'}(\hat{m}, \beta)$ tend to constants as β tends to 1.*

Proof: We examine the ratio between $\mathsf{P}_{1'}(\hat{m}, \beta)$ and $\mathsf{P}_{2'}(\hat{m}, \beta)$

$$\frac{\mathsf{P}_{1'}(\hat{m}, \beta)}{\mathsf{P}_{2'}(\hat{m}, \beta)} = \frac{2\beta}{(\hat{m} - 1)(1 - \beta)}.$$

Substituting $\beta_{\hat{m}}$ from (14) yields

$$\frac{\mathsf{P}_{1'}(\hat{m},\beta_{\hat{m}})}{\mathsf{P}_{2'}(\hat{m},\beta_{\hat{m}})}=\frac{\sqrt{2\hat{m}(\hat{m}-1)}}{\hat{m}-1},$$

which tends to a constant as $\hat{m}$ tends to infinity. The ratio and the sum both tending to constants imply that each of $\mathsf{P}_{1'}(\hat{m},\beta)$ and $\mathsf{P}_{2'}(\hat{m},\beta)$ tends to a constant. ■

B. Overhead analysis

While the greatest appeal of on-line fountain codes is in the presence of adversarial or other non-random losses, it is important to analyze them in the case where losses are random. The objective of the forthcoming analysis is to show that the on-line fountain codes proposed here have acceptable overheads, far below what existing on-line fountain codes require.
The analysis concentrates on the simplified code of Construction 2, because analyzing the optimal Construction 1 is challenging. It is clear that the overhead will only decrease if the optimal code is used instead of the simplified one.

We begin at the build-up phase of Section V-A. Randomly chosen edges added in the build-up phase construct a random graph $\mathcal{G}$. To analyze the properties of $\mathcal{G}$, we use known results on random graphs found in [1, Ch.10]. At the end of the build-up phase, $\mathcal{G}=G(k,p)$ is a random graph on k vertices, where each of the $k(k-1)/2$ possible edges is taken with probability p independently of other edges. We define $p=c/k$, and note the known relationship between c and β_0, the fractional size of the largest connected component, as

$$\beta_0+e^{-c\beta_0}=1. \tag{18}$$

Hence for each specified component size $\beta_0<1$, there is a unique density parameter $c>1$ that achieves it with high probability. The following results from [1, Ch.10] will be found useful in analyzing the completion phase.

Theorem 10. *[1] Given $c>1$ and $\beta_0<1$ with the relation given in* (18)*, in a random graph $\mathcal{G}=G(k,c/k)$ the remainder sub-graph outside the large component is itself a random graph $\mathcal{G}'=G(t,d/t)$, where $t=(1-\beta_0)k$ and $d=c(1-\beta_0)<1$.*

We will also use the following classical results by Erdős and Rényi.

Corollary 11. *[6] In a random graph $\mathcal{G}=G(k,c/k)$ with $c>1$, almost always all the components except the large component are of sizes $O(\log k)$. Furthermore, the number of small components that have cycles is vanishingly small.*

With the properties of $\mathcal{G}'$ quoted above, we turn to analyze the completion phase. We start with a qualitative description of the completion phase's dynamics, then move to more precise statements. At the outset of the completion phase, $\mathcal{G}'$ has small tree components (including isolated vertices, which are trivial trees). Case 1' symbols move components out of $\mathcal{G}'$ to the large (black) component, and Case 2' symbols add edges in $\mathcal{G}'$. Since $\mathsf{P}_{1'}(\hat{m},\beta)$ and $\mathsf{P}_{2'}(\hat{m},\beta)$ are both constants of the same order, a component of $\mathcal{G}'$ cannot grow much before it is colored black. This is because the probability to add an edge touching a given component is similar to the probability to connect the same component to the large component. Now given that the components of $\mathcal{G}'$ remain much smaller than a constant fraction of k, the probability to introduce a cycle in $\mathcal{G}'$ is negligible. Thus with high probability a Case 2' symbol is not redundant. The next theorem formally proves that with high probability every Case 2' symbol in the completion phase adds an innovating edge to $\mathcal{G}'$.

Theorem 12. *The probability that a cycle is introduced (by a Case 2' symbol) to a component of $\mathcal{G}'$ before the component joins the main component (by a Case 1' symbol) tends to zero as k goes to infinity.*

Proof: Let a random process take an event from $\{X,Y,*\}$ at each discrete-time instance i. For X and Y with respective probabilities P_X and P_Y, the probability that X occurs before Y is

$$\sum_{i=0}^{\infty}(1-P_X-P_Y)^iP_X=\frac{P_X}{P_X+P_Y}.$$

(Any number of $*$ events are allowed before X.) For a given component of $\mathcal{G}'$ with l vertices, we take X to be the event that a new code symbol is a Case 2' edge that creates a cycle in the component, and Y to be the event of a Case 1' symbol connecting the component to the large component. $*$ represents all other events caused by a new code symbol. Then we write the probability P_X as

$$P_X=\mathsf{P}_{2'}(m,\beta)\left(\frac{l}{(1-\beta)k}\right)^2.$$

The left multiplicand is the probability that the symbol is Case 2'; the right multiplicand is the probability that both ends of the $\mathcal{G}'$ edge fall on the size-l component. The right multiplicand can be interpreted as a conditional probability to doubly-hit a component given the symbol is a Case 2' symbol. Similarly, the probability P_Y is written as

$$P_Y=\mathsf{P}_{1'}(m,\beta)\frac{l}{(1-\beta)k}. \tag{19}$$

The left multiplicand is the probability that the symbol is Case 1'; the right multiplicand is the probability that the vertex of $\mathcal{G}'$ connected to the main component belongs to the size l component. The probability that X happens before Y is then

$$\frac{P_X}{P_X+P_Y}=\frac{\mathsf{P}_{2'}(m,\beta)}{\mathsf{P}_{2'}(m,\beta)+\frac{(1-\beta)k}{l}\mathsf{P}_{1'}(m,\beta)}. \tag{20}$$

It is clear that the expression in (20) tends to 0 as k tends to infinity, so long as $\mathsf{P}_{1'}(m,\beta)$ and $\mathsf{P}_{2'}(m,\beta)$ are both constants for any β, and in addition l is bounded from above by a function that is $o(k)$. The former is proved in Corollary 9; the latter is proved in the following lemma.

Lemma 13. *The probability that a $\mathcal{G}'$ component grows by more than $\log k$ vertices tends to zero as k goes to infinity.*

Proof: The proof follows similar lines to the main theorem's proof. For a given component of $\mathcal{G}'$ with l vertices, we now take Z to be the event that a new code symbol is a Case 2' edge that connects the component to a different component within $\mathcal{G}'$. We have

$$P_Z = \mathsf{P}_{2'}(m,\beta)\left[2\frac{l}{(1-\beta)k}\left[1-\frac{l}{(1-\beta)k}\right]\right] < \mathsf{P}_{2'}(m,\beta)\frac{2l}{(1-\beta)k}.$$

Y as before is the event of a Case 1' symbol connecting that component to the large component. Given P_Y in (19), the probability that Z occurs $\log k$ times before Y occurs is thus at most

$$\left(\frac{2\mathsf{P}_{2'}(m,\beta)}{2\mathsf{P}_{2'}(m,\beta)+\mathsf{P}_{1'}(m,\beta)}\right)^{\log k},$$

which clearly tends to zero assuming constant $\mathsf{P}_{i'}(m,\beta)$ probabilities for any β. ■

Since the components of $\mathcal{G}'$ at the beginning of the completion phase are of sizes $O(\log k)$ (Corollary 11), by Lemma 13 their sizes remain $O(\log k)$ throughout the completion phase. This shows that $l = o(k)$, which is sufficient for (20) to go to zero. This proves the theorem. ■

Theorem 12 implies that Case 2' symbols are not redundant with probability tending to 1, hence we have the following result.

Theorem 14. *The expected number of Case 1' and Case 2' symbols (combined) required to complete decoding in the completion phase is*

$$k(1-\beta_0)\left[1-\frac{(1-\beta_0)c}{2}\right], \tag{21}$$

in the limit of large k.

Proof: The number of symbols left to decode after the build-up phase equals t, the number of vertices in $\mathcal{G}'$. By the definition of the build-up phase we have

$$t = k(1-\beta_0). \tag{22}$$

According to Theorem 10, the expected number of edges in $\mathcal{G}'$ at the beginning of the completion phase is

$$\frac{1}{2}td = \frac{1}{2}k(1-\beta_0)(1-\beta_0)c. \tag{23}$$

A classical result by Erdős and Rényi states the following.

Lemma 15. *[6] In a random graph $\mathcal{G}' = G(t, d/t)$ with $d < 1$, the expected number of connected components equals*

$$t - \frac{td}{2} + O(1),$$

where the term $O(1)$ depends on d, but is a constant not growing with t.

Lemma 15 is implied by the well known fact that almost all of the vertices in a random graph with $d < 1$ are in connected components that are *trees*. Substituting t from (22) and $td/2$ from (23), the expected number of components in $\mathcal{G}'$ equals

$$k(1-\beta_0)\left[1-\frac{(1-\beta_0)c}{2}+o(1)\right], \tag{24}$$

where the term $o(1)$ tends to zero as k tends to infinity. By Proposition 1 the number of symbols the receiver needs to receive to fully decode equals the number of components in $\mathcal{G}'$, where we count a received symbol for that purpose only if it reduces the number of components by one. Case 1' symbols always reduce the number of components by one. By Theorem 12 all but a vanishing fraction of Case 2' symbols reduce the number of components by one. Hence the expected number of Case 1' and Case 2' symbols (combined) required in the completion phase equals to

$$\frac{k(1-\beta_0)\left[1-\frac{(1-\beta_0)c}{2}+o(1)\right]}{1-o(1)}.$$

This equals the expression in the theorem statement when k tends to infinity.

■

This leads to the main analysis results that now follow.

Theorem 16. *For any choice of $\beta_0 < 1$, the expected number N of code symbols required for decoding the k input symbols is bounded by*

$$N < \frac{1}{2}ck + \frac{e^{\sqrt{2}}}{1+\sqrt{2}}k(1-\beta_0)\left[1-\frac{1}{2}(1-\beta_0)c\right], \tag{25}$$

where c and β_0 are related by $\beta_0 + e^{-c\beta_0} = 1$.

Proof: The first term in (25) is the expected number of symbols received in the build-up phase. This is from random-graph theory [1, Ch.10], whereby reaching a component of size $\beta_0 k$ happens when a vertex has on average c edges (the factor $1/2$ is because each edge has two ends). The second term in (25) is the expected number of symbols received in the completion phase. The constant factor in the second term is the inverse of the lower bound on $\mathsf{P}_{1'}(m,\beta) + \mathsf{P}_{2'}(m,\beta)$ from Theorem 8, which is an upper bound on the expected number of received symbols per symbol of Case 1' or 2'. ■

Choosing $\beta_0 = 0.645$ for ending the build-up phase (which gives $c = 1.6$), we get the following corollary.

Corollary 17. *The expected redundancy overhead of Construction 2 is bounded by*

$$\frac{N-k}{k} < 0.236.$$

C. Discussion and comparison

It is important to note that the 0.236 upper bound on the expected overhead may be a substantial over-estimate of the true overhead. While the probabilities of Cases 1' and 2' are known for any β, our analysis was only able to incorporate the $\beta \to 1$ limit value of (12) as a lower bound. More involved arguments on random-graph dynamics may tighten this bound. In an experimental study of the simplified code we observe overheads smaller than 0.2, even for relatively short block lengths k. We also observe experimentally that the overhead is not very sensitive to the choice of β_0 to end the build-up phase. A complete experimental view of on-line fountain codes is deferred to a future publication.

Comparing the proven overhead upper bound of Construction 2 to known on-line schemes, a significant improvement is offered. Growth codes [8], the known state-of-the-art on-line fountain scheme, has an expected overhead that is bounded from below by $\ln 2 = 0.69$ (as explained in the sequel)– three times higher than the new proposal. The growth codes scheme can be seen as a special case of our simplified scheme, whereby the degrees are chosen to maximize the probability of Case 1' symbols alone. Therefore, degree 1 symbols are received until a $\beta_0 = 0.5$ fraction of black symbols exists (at which point symbol-decoding probability with degree 2 symbols crosses over that probability with degree 1 symbols). During that phase, the rate of symbol decoding is $d\beta = (1-\beta)dy$, where dy is an infinitesimal fraction of received symbols. Hence the fraction of received symbols y_0 required to obtain $\beta_0 = 0.5$ can be found by solving the simple integral

$$y_0 = \int_0^{0.5} \frac{d\beta}{(1-\beta)} = \ln 2.$$

After that phase, the probability of symbol decoding, for any degree m, is at most 0.5, thus the remaining 0.5 fraction of input symbols will need at least an additional 1 fraction of received symbols. Adding this 1 fraction to the $\ln 2$ fraction of the first phase gives a $\ln 2$ *lower* bound on overhead.

For the scheme of real-time oblivious codes [2] the authors quote an upper bound of 1 for the overhead, which is more than 5 times higher than Construction 2.

VI. Experimental Performance Evaluation

To complement the analytical evaluation of the new on-line fountain codes, in this section we conduct an experimental study through simulation. The study will concentrate on comparing the performance of the simplified code (Construction 2) with growth codes, the best previously known on-line fountain code. We also include in the study LT codes employing the robust-soliton degree distribution with the standard parameter choice $c = 0.9$, $\delta = 0.1$ (see e.g. in [7]). For Construction 2 we use the parameter $\beta_0 = 0.65$ to end the build-up phase. The purpose of this study is to illustrate the performance behavior of our codes with respect to the most similar (growth codes) and most understood (LT codes) counterparts. There are many other alternatives of fountain codes that employ feedback, which are likely to have better performance at many operation regimes. Given that each such alternative makes a different use of feedback (number of feedback transmissions, their distribution across the transmission block, the type of information they carry, etc.), we found it impossible to perform a fair comparison. Therefore, the subsequent results should be interpreted only as an illustrative demonstration of the performance achievable by codes with the on-line property, and *not* as a claim of superiority of the new codes over known fountain schemes with feedback in general.

A. Coding overhead

First in the performance figures to evaluate is the coding overhead. To measure the overhead, we simulate the coding schemes and count the number of code symbols needed to be received to complete decoding of the k input symbols. We repeat the simulation over many iterations, and plot the average number of required code symbols. Figure 5 shows the results of this experiment for input block lengths $k \in \{1000, 1500, 2000, 2500\}$, plotted for each of the two on-line coding schemes under comparison. It can be seen in

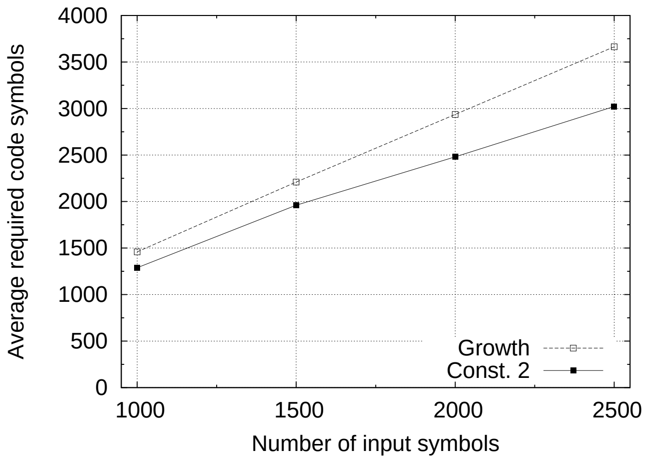

Figure 5. The average number of code symbols required at the receiver for full decoding. Solid line: Construction 2, dashed line: Growth codes.

Figure 5 that for the block lengths k used in the experiment, Construction 2 offers great performance advantage over Growth codes. The conclusion from this plot is that the refinement of the on-line property to also include degree-2 symbols pays nicely in total overhead.

B. Feedback cost

In the previous sub-section we compared the benefits of on-line fountain codes in reducing communication overhead from the sender to the receiver. Now we want to evaluate the cost of these benefits in feedback transmissions from the receiver back to the sender. To measure feedback cost, we added to the same simulation of Section VI-A a counter of feedback transmissions required by the two on-line fountain codes. A feedback transmission is required by the on-line fountain code (Growth codes and Construction 2) whenever the respective optimal degree changes following the receipt of the previous code symbol. Hence for this experiment we assume that finding the optimal degree given the decoding state is performed by the receiver, thus allowing to issue a feedback transmission only when there is a change in the optimal value. Following this experiment we plot in Figure 6 the average numbers of feedback transmissions for $k \in \{1000, 1500, 2000, 2500\}$. The results of Figure 6 reveal that the use of feedback by Construction 2 is limited to only 2-3% of the block length, and that the feedback cost is roughly half that of Growth codes. Therefore, Construction 2 is better than Growth codes in both overhead and feedback. At this point it is important to note that these results are only a starting point for a very interesting possibility to refine the current on-line fountain codes such that overhead and feedback are jointly minimized. This interesting direction is deferred to future work.

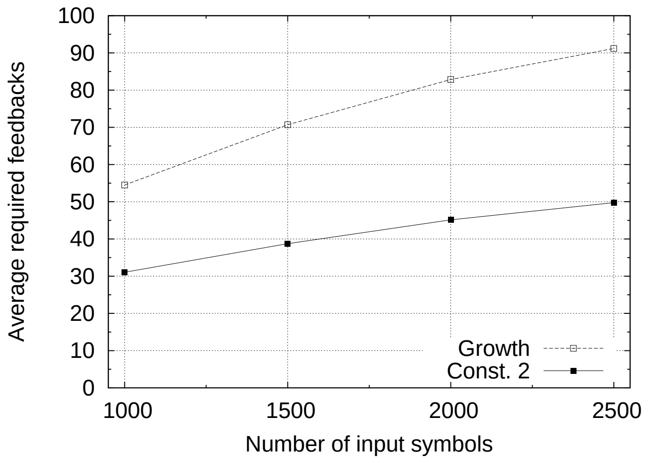

Figure 6. The average number of feedback transmissions per code block. Solid line: Construction 2, dashed line: Growth codes.

C. Intermediate symbol decoding

The previous sub-sections showed results pertaining to the *block* decoding performance. Now we want to "zoom into" the decoding process, and see for each intermediate number of received code symbols how many of the input symbols are decoded. There are two purposes for measuring the intermediate-decoding performance. One is to shed quantitative light on the way the decoding state evolves through the decoding process. Another is to evaluate the *real-time* performance of the codes, which amounts to the ability of the decoder to pass decoded symbols over to the application before the full-block decoding is completed. Results comparing the intermediate-decoding performance of the three codes are given in Figure 7. The plots show the median results among 100 runs, thus capturing the behavior of the majority of the coding instances. There

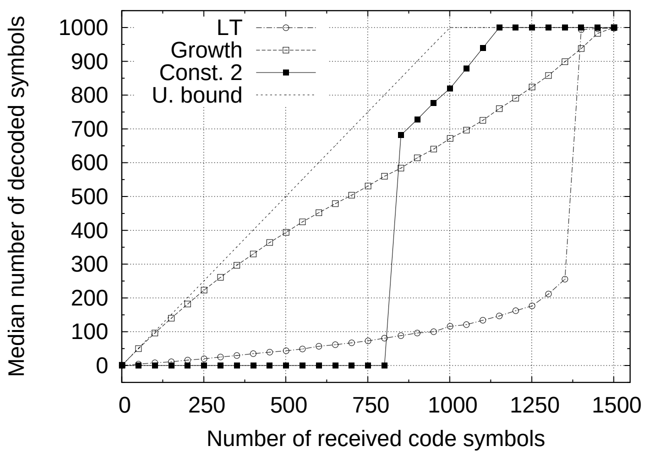

Figure 7. For $k = 1000$ the median number of decoded input symbols for each number of received code symbols. Solid line: Construction 2, dashed line: Growth codes, dash-dotted line: LT codes, no-markers line: upper bound.

are several interesting facts to observe in Figure 7. First to see is the initial flat interval in the decoding curve of Construction 2. This flatness stems from the build-up phase where only degree 2 symbols are sent, and thus no symbol is decoded. Next, after the build-up phase is over, the number of decoded symbols jumps, and from then on continues at a roughly linear slope to complete first from the three codes. It is also interesting to link the features of the Construction 2 curve to the random-graph analysis of Section V-B. The x-coordinate at which the jump occurs corresponds to the threshold β_0, and specifically to the number of symbols required to be received until the formation of a connected component of size $\beta_0 k$. The magnitude of the jump amounts to a little more than $\beta_0 k$ symbols, including both the large component and other smaller components decoded with the first received symbols in the completion phase. Finally, the slope at the latter part of the curve shows good real-time performance in the completion phase. That is to say that optimizing the degrees for both $\mathsf{P}_{1'}$ and $\mathsf{P}_{2'}$ does not hurt the intermediate-decoding rate. Clearly there is an interesting tradeoff here between minimizing the full-decoding time and maximizing the real-time performance of the code. We can use β_0 as a simple "knob" to control this tradeoff (reducing β_0 will shrink the flat interval, but will result in a smaller jump and a slower completion slope).

VII. Conclusion

In addition to providing low-overhead on-line fountain constructions, the results of this paper open new avenues for future research. The most natural direction is attempting to further reduce the overhead, and at the same time deriving lower bounds on the overhead required to attain the on-line property. Another important extension of this work is to address point-to-multipoint communications, where the encoder strategy needs to be optimized for multiple receivers simultaneously. Finally, it is also important to consider the effect on the on-line property when the feedback of decoding-state information from the receiver to the sender is limited.

VIII. Acknowledgment

We would like to thank Morteza Hashemi for providing code that helped obtaining the simulation results. We also want to thank the anonymous referees for suggesting significant additions to the paper, which improved it considerably.

References

[1] N. Alon and J. Spencer, *The probabilistic method*. Wiley, 2000.

[2] A. Beimel, S. Dolev, and N. singer, "RT oblivious erasure correcting," *IEEE/ACM-Trans-Networking*, vol. 15, no. 6, pp. 1321–1332, Dec. 2007.

[3] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proc. ACM SIGCOMM'98*, Vancouver BC, Canada, 1998, pp. 56–67.

[4] Y. Cassuto and A. Shokrollahi, "On-line fountain codes for semi-random loss channels," in *Proc. IEEE Information Theory Workshop*, Paraty, Brazil, 2011.

[5] J. Considine, "Generating good degree distributions for sparse parity check codes using oracles," CS Department, Boston University, Tech. Rep. BUCS-TR-2001-019, 2001.

[6] P. Erdős and A. Rényi, "On the evolution of random graphs," in *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, 1960, pp. 17–61.

[7] A. Hagedorn, S. Agarwal, D. Starobinski, and A. Trachtenberg, "Rateless coding with feedback," in *INFOCOM 2009, IEEE*, 2009, pp. 1791–1799.

[8] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, “Growth codes: maximizing sensor network data persistence,” in *Proc. ACM SIGCOMM’06*, New York NY USA, 2006, pp. 255–266.

[9] S. Kokalj-Filipovic, P. Spasojevic, E. Soljanin, and R. Yates, “ARQ with doped fountain decoding,” in *Spread Spectrum Techniques and Applications, 2008. ISSSTA ’08. IEEE 10th International Symposium on*, 2008, pp. 780–784.

[10] M. Luby, “LT codes,” in *Proc. of the Annual IEEE Symposium on Foundations of Computer Science FOCS*, Vancouver BC, Canada, 2002, pp. 271–280.

[11] A. Shokrollahi, “Raptor codes,” *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[12] A. Talari and N. Rahnavard, “LT-AF codes: LT codes with alternating feedback,” in *Proc. of the IEEE International Symposium on Info. Theory*, Istanbul, Turkey, July 2013, pp. 2646–2650.

Yuval Cassuto (S’02-M’08-SM’14) is a faculty member at the Department of Electrical Engineering, Technion – Israel Institute of Technology. His research interests lie at the intersection of the theoretical infomration sciences and the engineering of practical computing and storage systems.

During 2010-2011 he has been a Scientist at EPFL, the Swiss Federal Institute of Technology in Lausanne. From 2008 to 2010 he was a Research Staff Member at Hitachi Global Storage Technologies, San Jose Research Center. From 2000 to 2002, he was with Qualcomm, Israel R&D Center, where he worked on modeling, design and analysis in wireless communications.

He received the B.Sc degree in Electrical Engineering, summa cum laude, from the Technion, Israel Institute of Technology, in 2001, and the MS and Ph.D degrees in Electrical Engineering from the California Institute of Technology, in 2004 and 2008, respectively.

Dr. Cassuto has won the 2010 Best Student Paper Award in data storage from the IEEE Communications Society, as well as the 2001 Texas Instruments DSP and Analog Challenge $100,000 prize.

Amin Shokrollahi (M’00-SM’06-F’07) has worked and published on a variety of topics, including coding theory, computational number theory and algebra, and computational/algebraic complexity theory. He is best known for his work on iterative decoding algorithms of graph based codes, an area in which he has published several influential papers, and holds more than 20 granted and pending patents. He is the co-inventor of Tornado codes, and the inventor of Raptor codes. His codes have been standardized and successfully deployed in industrial applications involving data transmission over lossy networks.

Amin finished his Ph.D. in 1991 at the University of Bonn. From 1995 to 1998, he was a Senior Researcher at the International Computer Science Institute in Berkeley. From 1998 to 2000, he was a Member of the Technical Staff at the Mathematical Sciences Research Center at Bell Laboratories. In 2000, he became the Chief Scientist of Digital Fountain, a company specializing on fast and reliable data transmission on unreliable networks. He held this position until early 2009, when the company was acquired by Qualcomm. In 2003, Amin joined the faculty of EPFL where he holds a position as a full professor jointly in the departments of Mathematics, and of Computer Science. He is the co-founder of Kandou Technologies, a company specializing in the design and implementation of high speed and energy efficient serial links of which is he currently the CEO.

Amin is a Fellow of the IEEE. He was awarded the best paper award of the IEEE IT Society in 2002 for his work on iterative decoding of LDPC codes, the IEEE Eric Sumner Award in 2007 for the development of Fountain Codes, and the joint Communication Society/Information Theory Society best paper award of 2007 for his paper on Raptor Codes. He is also a recipient of the prestigious 2009 Advanced Research Grant of the European Union Research Council. In addition, he is the co-recipient of the 2012 IEEE Hamming medal.