On-Line Fountain Codes for Semi-Random Loss Channels

Yuval Cassuto and M. Amin Shokrollahi
EPFL – School of Computer and Communication Sciences
ALGO Laboratory
1015 Lausanne, Switzerland
{yuval.cassuto,amin.shokrollahi}@epfl.ch

***Abstract*— A fountain coding framework is proposed that endows receivers with the ability to monitor and control the decoding progress given the instantaneous network conditions. These on-line features allow an optimal recovery from losses manifested by adversarial or other not purely random processes. A uni-partite graph structure and accompanying algorithms are used to efficiently calculate optimal instantaneous degrees. Using analysis of random-graph processes, the average overhead of a simplified scheme is shown to be upper bounded by 0.236, significantly lower than any known on-line fountain scheme.**

I. Introduction

Fountain coding was proposed [2] 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 [5], and later improved in complexity [6]. 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 predefined 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. While this unavoidably requires some feedback from the receivers back to the sender, an effective on-line scheme can control the size and frequency of feedback transmissions, and trade them off with redundancy overhead.

In this paper we propose a fountain coding framework that offers a good balance between overhead and on-line capabilities. Known on-line schemes, such as [4] and [3], aggressively minimize decoding latency, and as a consequence have very high overheads. In our scheme, rather than equating the instantaneous decoding progress with symbol decoding events, we consider a more general measure of progress. To allow that, the instantaneous decoding state of the receiver is tracked by a *uni-partite* graph data structure, which can be used to maximize the decoding progress given the current state and receiver preferences. In this way, it is possible for a receiver to optimally recover from even the most unfortunate (or even adversarial) backdrop of losses.

The primary purpose of this paper is to detail the fundamental elements of the framework, namely the uni-partite graph structure (Section II), efficient determination of instantaneously optimal degrees (Section III), and a simplified scheme (Section IV), which is analytically shown to have a 0.236 upper bound on its expected overhead – better than any on-line fountain scheme that we are aware of. These results enable and motivate employing on-line fountain codes in real networks, but the details of their actual realizations are deferred to future publications.

II. Fountain Codes on Uni-partite Graphs

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, *output nodes*, represent the output code symbols. An edge between an input node x_i and an output node y_j marks that input symbol x_i is one of the summands[1] of output 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 output nodes are 2 or less, the

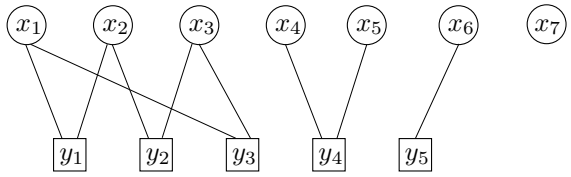

Figure 1. A fountain code as a bi-partite graph

code 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 with an edge if there is an output 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). In the example, symbol x_6 is black thanks to the output symbol $y_5 = x_6$.

This work was supported by a grant from the European Research Council.

[1] symbol additions are assumed to be modulo 2 throughout the paper

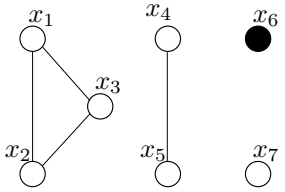

Figure 2. A fountain code as a uni-partite graph

A. Implicit edges and connected components

In addition to edges representing explicit output 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 output 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 progress is fully described by the connected components of the graph, as well as 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.). Another simple observation is that an incoming edge between a black node and a white node colors in black all the nodes in the white node's connected component. For example, a symbol $x_4 + x_6$ would decode x_4 by subtracting the known x_6, followed by decoding x_5 by subtracting x_4 from $x_4 + x_5$.

Suppose the code graph at a given instance 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 clearly

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

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

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

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

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

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.

Let $\mathcal{B} \triangleq \mathcal{A}_{\downarrow t}$ be the polynomial with coefficients specified as

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

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

B. High-degree symbols with reduced degree up to 2

While useful to provide a succinct description of the decoding state, the uni-partite graph model only permits recording sum relationships of degree 2. It is clear that a fountain code with no output degrees greater than 2 is unusable due to extreme inefficiency. Hence a key primitive of the proposed coding scheme is to receive high degree symbols, and reduce them to degrees 1 or 2 representable by the uni-partite graph model. In particular, an important undertaking for the receiver is to calculate output-symbol degrees that maximize the probability of successful degree reduction given the *instantaneous decoding state*.

III. Instantaneously Optimal Degrees

In the uni-partite graph framework, decoding progress is achieved if an incoming output symbol is reduced to one of the following

1) A connected component turning black
2) A connecting edge between two different 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 between 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 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 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.

We wish to count the number of degree m output symbols that result in Case 1 or Case 2 symbols. Assuming the degree m output symbol is drawn uniformly at random, dividing this number by the total number of choices $\binom{k}{m}$ will give the probability of achieving decoding progress. Formalizing the above, given a component enumerator $\mathcal{A}(x)$, let $N_1(m, \mathcal{A})$ denote the number of degree m output symbols that color a component in black (Case 1). Let $N_2(m, \mathcal{A})$ similarly denote the number of degree m output symbols that connect two different components (Case 2).

We start off with counting Case 1 combinations. Let $\mathsf{OEs}(\mathcal{U}, j)$ be the number of distinct combinations of j symbols that have an odd number of symbols in one component, and even numbers (including zero) symbols in all other components, given a component enumerator $\mathcal{U}(x)$. The symbol OEs stands for *Odd-Evens*. Then we can write $N_1(m, \mathcal{A})$ as

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

where s in the summation is the number of black symbols in the degree m output symbol. In turn, $\mathsf{OEs}(\mathcal{U}, j)$ can be

written as

$$\mathsf{OEs}(\mathcal{U},j)=\sum_{\substack{i=1\\ i\text{ odd}}}^{j}\sum_{t=i}^{k}\binom{t}{i}A_t\mathsf{Es}(\mathcal{U}_{\downarrow t},j-i,1) \quad (1)$$

i in the outer sum is the odd number of symbols in the component to turn black; t in the inner sum is the size of this component. The function $\mathsf{Es}(\mathcal{V},l,1)$ counts the number of distinct combinations of l symbols that have even numbers (including zero) of symbols in all components, given a component enumerator $\mathcal{V}(x)$ (Es stands for *Evens*). The function Es is invoked on the polynomial $\mathcal{U}_{\downarrow t}$, to exclude the component chosen to have an odd number of symbols. Calculating $\mathsf{Es}(\mathcal{V},l,1)$ can be done efficiently by evaluating the following recursive formula with $d=1$

for $l>0$

$$\mathsf{Es}(\mathcal{V},l,d)=\frac{1}{d}\sum_{\substack{i=2\\ i\text{ even}}}^{l}\sum_{t=i}^{k}\binom{t}{i}A_t\mathsf{Es}(\mathcal{V}_{\downarrow t},l-i,d+1)$$

for $l=0$

$$\mathsf{Es}(\mathcal{V},0,d)=1$$

the parameter d input to Es specifies the *depth* of the evaluation, and its purpose is to cancel multiple-counting of combinations that differ only by the order in which the components are chosen.

Moving to count Case 2 combinations, we define $\mathsf{OOEs}(\mathcal{A},j)$ to be the number of distinct combinations of j symbols that have odd numbers of symbols in *two* 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*. Then we can write $N_2(m,\mathcal{A})$ as

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

s, as before, is the number of black symbols in the degree m output symbol. Now we write $\mathsf{OOEs}(\mathcal{A},j)$ as

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

i in the outer sum is the odd number of symbols in the first component; 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 $\mathsf{OEs}(\mathcal{U},l)$ from (1) can be used. The $1/2$ factor cancels a 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 denote by $\mathsf{P}_1(m,\mathcal{A})$ and $\mathsf{P}_2(m,\mathcal{A})$ the probabilities that a degree m output symbol results in Case 1 and Case 2 events, respectively. Then for $i\in\{1,2\}$

$$\mathsf{P}_i(m,\mathcal{A})=\frac{N_i(m,\mathcal{A})}{\binom{k}{m}}$$

and to maximize the probability of decoding progress we pick m to be

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

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 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*

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

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
$\mathsf{P}_1+\mathsf{P}_2$	0.75	0.714	0.75	**0.8**	0.75	0.714	0.75	0

TABLE I
DECODING-PROGRESS PROBABILITIES FOR DIFFERENT DEGREES m

IV. A Simplified On-Line Fountain Scheme

In the previous section, it was shown that receiving clients can efficiently calculate the degree that maximizes the probability of decoding progress at any step of decoding. In this section, we sacrifice the optimality and consider a simplified on-line fountain scheme that is simpler both for implementation and for analysis. In the analysis part of this section, it is shown that this simple scheme has a reasonably low redundancy overhead, and that it significantly outperforms known on-line fountain schemes.

A. Specification

In the simplified scheme, the decoding is first coarsely divided to two phases called *build-up* and *completion*. In the build-up phase the degree is 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. A full specification of the two-phase decoding procedure now follows.

1) Build-up

At the build-up phase, the client receives output symbols of degree 2. These symbols add edges to the decoding graph. The build-up phase continues until a connected component of size $|B| = \beta_0 k$ exists in the graph ($0 < \beta_0 < 1$ is a parameter). Then degree-1 symbols are received until a symbol in the large component is received, which colors the symbols in the large component black.

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, 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 III is the following. The appeal of cases 1' and 2' is that their probabilities can be calculated based on β alone, without need to know the complete component enumerator (as detailed in the next paragraph). 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 III due to the low probability of multiple symbols in the same component.

The two cases 1' and 2' at the completion phase are illustrated in Figure 3.

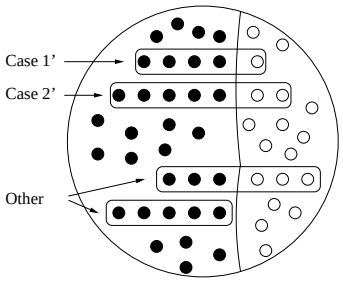

Figure 3. Case 1' and Case 2' sought in the completion phase. Other cases (showed below) result in discarding the received symbol.

During the completion phase, degrees need to be chosen to maximize the probability of decoding progress, as expressed by cases 1' and 2'. We now turn to calculate $\mathsf{P}_{1'}(m,\beta)$ and $\mathsf{P}_{2'}(m,\beta)$, the probabilities of Case 1' and Case 2', respectively. Note that $\mathsf{P}_{1'}$ and $\mathsf{P}_{2'}$ depend on m and β, the fraction of black symbols. This is in contrast to P_1 and P_2 of Section III, which depend on m and the full component enumerator.

$$\begin{aligned}\mathsf{P}_{1'}(m,\beta) &= \binom{m}{1}\beta^{m-1}(1-\beta)\\ \mathsf{P}_{2'}(m,\beta) &= \binom{m}{2}\beta^{m-2}(1-\beta)^2\end{aligned}$$

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

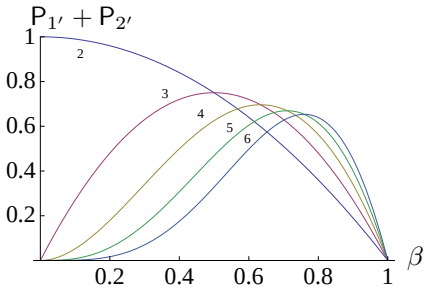

Figure 4. The sum probability of cases 1' and 2' as a function of the fraction of black symbols. Plotted for $m = 2,3,4,5,6$.

Theorem 1. *For any β, there exists a $\hat{m}$ such that*

$$\mathsf{P}_{1'}(\hat{m},\beta)+\mathsf{P}_{2'}(\hat{m},\beta) > \frac{21}{8}e^{-3/2} = 0.5857 \qquad (2)$$

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

$$\frac{2m-3}{2m} \leqslant \beta < \frac{2m-1}{2m+2}$$

B. Overhead analysis

While the motivation to consider on-line fountain codes comes from adversarial or non-random losses, it is important to analyze them in the case where losses are random. Since real-world channels are somewhere in between random and adversarial channels, it is important to show that these codes have an acceptable overhead even when the loss channel is purely random.

The analysis begins at the *build-up* phase of sub-section IV-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. We define $p = c/k$, and note the known relationship between c and β_0, the fractional size of the largest connected component, as $\beta_0 + e^{-c\beta_0} = 1$. Hence for each specified component size $\beta_0 < 1$, there is a unique density parameter $c > 1$ that achieves it with high probability. At that point when $\mathcal{G} = G(k, c/k)$, the remainder sub-graph outside the large component has $(1-\beta_0)k$ vertices, and all its components are of size $O(\log k)$. More precisely, the remainder sub-graph, denoted $\mathcal{G}'$, is itself a standard random graph $\mathcal{G}' = G(t, d/t)$, where $t = (1-\beta_0)k$ and $d = c(1-\beta_0) < 1$.

With the properties of $\mathcal{G}'$ described 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 close to zero. Thus no Case 1' or 2' symbol is 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 2. *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

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

(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 output symbol is a Case 2' edge addition 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 output 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. Similarly, the probability P_Y is written as

$$P_Y = \mathsf{P}_{1'}(m,\beta)\frac{l}{(1-\beta)k} \tag{3}$$

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

$$\frac{P_X}{P_X+P_Y} = \frac{\mathsf{P}_{2'}(m,\beta)}{\mathsf{P}_{2'}(m,\beta)+\frac{(1-\beta)k}{l}\mathsf{P}_{1'}(m,\beta)} \tag{4}$$

It is clear that the expression in (4) tends to 0, 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 latter is proved in the following lemma.

Lemma 3. *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 output symbol is a Case 2' edge addition that connects the component to a different component within $\mathcal{G}'$. Y as before is the event of a Case 1' symbol connecting the component to the large component. Now we have

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

Given P_Y in (3), 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 $O(\log k)$ [1], by Lemma 3 their sizes remain $O(\log k)$ throughout the completion phase. This shows that $l = o(k)$, which is sufficient for (4) to go to zero. This proves the theorem. ■

The conclusion from Theorem 2 is that the expected number of Case 1' and Case 2' symbols (combined) required to complete decoding is

$$k(1-\beta_0) - \frac{1}{2}td = k(1-\beta_0)\left[1-\frac{1}{2}(1-\beta_0)c\right]$$

where $k(1-\beta_0)$ is the number of symbols left to decode (the number of vertices in $\mathcal{G}'$) and $\frac{1}{2}td$ is the number of edges in $\mathcal{G}'$, both at the beginning of the completion phase. This leads to the main analysis results that now follow.

Theorem 4. *The expected number of output symbols N required for decoding the k input symbols is bounded by*

$$N < \frac{1}{2}ck + \frac{8}{21}e^{3/2}k(1-\beta_0)\left[1-\frac{1}{2}(1-\beta_0)c\right]$$

the first and second terms are the expected numbers of symbols received in the build-up and completion phases, respectively. c and β_0 are related by $\beta_0 + e^{-c\beta_0} = 1$; the multiplicative constant in the second term is the inverse of the lower bound on the case 1' and 2' combined probability from (2). Choosing $\beta_0 = 0.645$ for ending the build-up phase (which gives $c = 1.6$), we get the following corollary.

Corollary 5. *The expected redundancy overhead of the on-line coding scheme 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 $\beta \to \infty$ limit value of (2) as a lower bound. More involved arguments on random-graph dynamics may tighten this bound, and evidence to its true value may be obtained by experimental study.

Comparing the proven overhead of our on-line fountain scheme to known on-line schemes, a significant improvement is offered. Growth Codes [4], the known state-of-the-art on-line fountain scheme, has an expected overhead that is bounded from below by $\ln 2 = 0.69$ (proof omitted) – three times higher than the new proposal.

REFERENCES

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

[2] 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.

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

[4] 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.

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

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