Data Persistence in Large-scale Sensor Networks with Decentralized Fountain Codes

Yunfeng Lin, Ben Liang, Baochun Li

***Abstract*—It may not be feasible for sensor networks monitoring nature and inaccessible geographical regions to include powered sinks with Internet connections. We consider the scenario where sinks are not present in large-scale sensor networks, and unreliable sensors have to collectively resort to storing sensed data over time on themselves. At a time of convenience, such cached data from a small subset of live sensors may be collected by a centralized (possibly mobile) collector. In this paper, we propose a decentralized algorithm using fountain codes to guarantee the persistence and reliability of cached data on unreliable sensors. With fountain codes, the collector is able to recover all data as long as a sufficient number of sensors are alive.**

We use random walks to disseminate data from a sensor to a random subset of sensors in the network. Our algorithms take advantage of the low decoding complexity of fountain codes, as well as the scalability of the dissemination process via random walks. We have proposed two algorithms based on random walks. Our theoretical analysis and simulation-based studies have shown that, the first algorithm maintains the same level of fault tolerance as the original centralized fountain code, while introducing lower overhead than naive random-walk based implementation in the dissemination process. Our second algorithm has lower level of fault tolerance than the original centralized fountain code, but consumes much lower dissemination cost.

I. Introduction

Wireless sensor networks consist of unreliable and energy-constrained sensors communicating with one another wirelessly. It has been a conventional assumption that, in wireless sensor networks, measured data in individual sensors are gathered (via data aggregation) and processed *en masse* at powered *sinks* with Internet connections. There are, however, at least two cases in which this assumption may not realistically hold. First, if sensor networks consist of a large number of sensors (in the order of tens of thousands or higher), it may not be energy efficient to gather measured data from sensors to sinks using data aggregation. Second, if sensors are randomly deployed in inaccessible geographical regions or environments, it may not be feasible to deploy powered sinks as well.

As we challenge the assumption that powered sinks with Internet connections exist in wireless sensor networks, our proposed vision is to ask the sensors to collaboratively store measured data over a historical period of time on themselves. At a later time of convenience, a (possibly mobile) *collector* (*e.g.*, a motor vehicle) collects such historical measured data directly from the sensors. In other words, rather than a "*push*" model in which sensors proactively send data periodically to the sink, we employ a "*pull*" model, where sensors are passively polled by the collector.

The authors are affiliated with the Department of Electrical and Computer Engineering, University of Toronto. Their email addresses are {*ylin,bli*}*@eecg.toronto.edu* and *liang@comm.utoronto.ca*.

Considering that all sensors are inherently unreliable and vulnerable to failures, the following question naturally arises: How do we reliably retrieve historical data that the sensors have gathered, even after a subset of them has failed? Intuitively, the sheer number of inexpensive sensors in the network provides natural fault tolerance, and if used wisely, leads to the desired persistence of measured data for a period of time. However, what is the algorithm that provides the highest degree of fault tolerance and data persistence?

In this paper, we propose a novel *decentralized implementation* of fountain codes in sensor networks, such that data can be encoded in a completely distributed fashion. In our proposed algorithms, a sensor disseminates its data to a random subset of sensors in the network, while each sensor only encodes data that it has received. As the collector collects a sufficient number of encoded data blocks by visiting more and more sensors, it is able to decode all original data with an efficient decoding process designed for fountain codes. The salient and original contribution of this paper is a solution to disseminate data from one sensor to others in an efficient and scalable fashion. As has been well known, conventional shortest-path routing algorithms require each sensor to maintain a routing table with a size proportional to the total number of sensors in the network. Alternatively, geographic routing protocols [1] are more scalable, with the assumption that sensors know their respective locations. Be that as it may, we have observed that our decentralized implementation of fountain codes does *not* require the support of a generic layer of routing protocols. Instead, we use *random walks* to disseminate data from one sensor to a random subset of sensors in the network. The salient advantage of random walks is that they only need local information, and do not assume the knowledge of sensor locations.

As far as we are aware, there is no existing study on distributed implementation of fountain codes through stateless random walks. Kamra *et al.* [2] have proposed *Growth codes*, a variant of LT codes [3] to maximize the amount of recovered data at the sink by increasing the code rate over time. However, our work addresses the data persistence problem when no sink is available. Furthermore, Growth codes require approximate time synchronization in the network to support identical speeds of code rate increases on all sensors, whereas our work does not have such requirement. Dimakis *et al.* [4] have shown a decentralized implementation of fountain codes, which is, to the best of our knowledge, the closest to our work. However, their implementation uses geographic routing and requires every node to know its location, whereas our algorithms do not have such assumption.

The remainder of this paper is organized as follows. Sec. II reviews fountain codes and random walks. Sec. III introduces our main algorithm towards using decentralized fountain codes

to improve data persistence in sensor networks without sinks. Sec. IV discusses the optimality of encoding multiple source blocks into one encoded block at each receiving node. The performance of our algorithms is evaluated extensively in Sec. V. We conclude the paper in Sec. VI.

II. Preliminaries

Using an analogy to *Redundant Array of Independent Disks* (RAID) in computer systems [5], we explain a number of alternatives that motivate the credo of this paper.

A. Why Fountain Codes?

Similar to RAID 1 disk arrays (mirrored disks), the first alternative is to utilize other sensors to "*mirror*" or "*replicate*" data from a particular sensor. After such mirroring, when a sensor fails, the data it has gathered before its failure can be retrieved from other "backup" sensors, improving data persistence and fault tolerance. As an example of existing work, Hass *et al.* [6] have addressed the fault tolerance of distributed location databases by using replication. It is easy to see that a large number of replicas are required to maintain a certain degree of fault tolerance, when the failure rate of sensors increases.

Analogous to RAID 6 disk arrays (in which Reed-Solomon codes are used), *error-correcting codes* may be used to alleviate the disadvantages of mirroring data. It has also been proposed that Reed-Solomon and LDPC codes may be used towards building reliable and distributed data storage systems over wide-area networks [7], [8]. There is one catch, however, in that the encoding and decoding processes of conventional error-correcting codes have to be implemented in a *centralized* fashion. To make good use of them in sensor networks, all data have to be delivered to a centralized sensor node, which encodes them and re-distributes the encoded data within the network. This is obviously not realistic, in that (1) a single sensor is not assumed to have sufficient energy or computational power to encode all the data received; (2) it suffers from a single point of failure; and (3) the communication cost of forwarding all data to a central node for encoding is prohibitive.

In wireless sensor networks, where data are generated from different sensors in a distributed way, it has been proposed in existing work that *random linear codes* may be used to improve the degree of fault tolerance [9], [10], [11] by decentralized encoding. Random linear codes are also investigated in distributed networked storage [12]. Unfortunately, similar to Reed-Solomon codes, the decoding process of random linear codes is computationally expensive, with a decoding complexity of $O(K^3)$, where K is the number of data blocks[1] to be encoded.

We believe that the superior decoding complexity of *fountain codes* — $O(K \ln K)$ — may come to our rescue, even though the encoding process of fountain codes is also centralized in coding literature. We briefly introduce LT codes [3], the first fountain codes proposed in the coding literature. We use LT codes in our subsequent discussions and performance evaluation. Fountain codes are *rateless* in the sense that the number of encoded data blocks that can be generated from the original data blocks (referred to as *source blocks hereafter*) is potentially unlimited, and the encoded blocks can be computed on the fly. Therefore, fountain codes are especially suitable for erasure channels where the erasure probabilities are unknown. Fountain codes also enjoy excellent computational efficiency for both encoding and decoding processes.

[1] A *data block* typically consists of one measurement sample on a sensor.

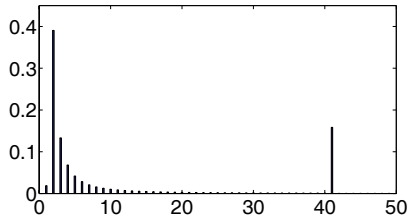

Fig. 1. Robust Soliton distribution for the case $K = 10000$, $c = 0.2$, and $\delta = 0.05$. Note that a *spike* exists at $K/R = 41$. Such a spike is sufficient replacement for all higher degrees. The encoded blocks with a degree higher than K/R are not essential in decoding.

In LT codes, it has been shown that K source blocks can be decoded from any subset of $K + O(\sqrt{K}\ln^2(K/\delta))$ encoded blocks with probability $1-\delta$. The encoding and the decoding complexity are both $K\ln(K/\delta)$. The number of source blocks that are used to generate an encoded block is referred to as the *degree* of the encoded block. The degree distribution of encoded blocks in LT codes follows the *Robust Soliton distribution*. We first define the *Ideal Soliton distribution* $\rho(\cdot)$:

$$\rho(i) = \begin{cases} 1/K & \text{if } i = 1, \\ 1/i(i-1) & \text{for } i = 2, 3, \ldots, K. \end{cases}$$

Then we let $R = c\ln(K/\delta)\sqrt{K}$ for some constant $c > 0$ and

$$\tau(i) = \begin{cases} R/iK & \text{for } i = 1, \ldots, K/R - 1 \\ R\ln(R/\delta)/K & \text{for } i = K/R \\ 0 & \text{for } i = K/R+1, \ldots, K \end{cases}.$$

The Robust Soliton distribution is defined as

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta}, \tag{1}$$

where $\beta = \sum_i \rho(i) + \tau(i)$. Fig. 1 shows an example.

The encoder of LT codes generates each encoded block independently by first randomly choosing the degree d of an encoded block from the Robust Soliton distribution, and then choose d distinct blocks from K source blocks uniformly at random. The encoded block is the bitwise exclusive-or (XOR) of the d source blocks. The decoding process utilizes the Belief Propagation (BP) algorithm, which intuitively is more computationally efficient than the general matrix inversion process, since degree-one encoded blocks are used to activate a cascading backward cancellation process, producing more degree-one encoded blocks.

B. Random Walks on Graphs

We describe random walks in the context of disseminating a source block, with sensors as the nodes in the graph. Given the source node n_0 of the block, the next hop n_1 is randomly chosen from the neighbors of n_0; when the block arrives at

n_1, the next hop n_2 is randomly chosen from the neighbors of n_1, and so on.

Since the choice of the next hop depends on only the current node, a random walk corresponds to a time-reversible Markov chain. The states of the Markov chain are the nodes in the graph. If the graph is ergodic (which is almost certain for the graph corresponds to the topology of a randomly deployed sensor network), the Markov chain has a steady-state distribution which equals to its limiting distribution. Therefore, if the length of the random walk is sufficiently long, the distribution (limiting distribution) of the random walk stopping at a particular node converges to the steady-state distribution.

The length of the random walk is proportional to the transmission cost of disseminating a source block, which must be minimized. The minimal length of a random walk to approximate the steady-state distribution within a certain error is called the *mixing time* of the random walk. The mixing time of random walks can be reduced by adjusting the transition matrix of the Markov chain. Boyd *et al.* [13] find the fastest mixing time of a random walk on a graph by solving an optimization problem. If their algorithm can be amenable to a distributed implementation, it is complementary to our decentralized implementation of fountain codes to be subsequently proposed. Boyd *et al.* [14] also show that natural random walks have the same asymptotic order of mixing time as the fastest random walk on Random Geometric Graphs [15], a popular topological model for multi-hop wireless networks. In this paper, we choose a variant of the Metropolis algorithm [16], which is a generalization of the natural random walks for the Markov chain with a non-uniform steady-state distribution.

The Metropolis algorithm computes the transition matrix, $P = [P_{ij}]$, of a time-reversible Markov chain, given its steady-state distribution $\pi = (\pi_1, \pi_2, \ldots)$ [16]:

$$P_{ij} = \begin{cases} \min(1, \pi_j/\pi_i)/D_m & \text{if } i \neq j \text{ and } j \in N(i), \\ 0 & \text{if } i \neq j \text{ and } j \notin N(i), \\ 1 - \sum_{j \neq i} P_{ij} & \text{if } i = j. \end{cases} \quad , \quad (2)$$

where $N(i)$ denotes the neighbors of node i and D_m denotes the maximal node degree in the graph. Clearly, the Metropolis algorithm is distributed, since each node only needs the steady-state probabilities of its neighbors to calculate the transition matrix. The transition probability entries in the transition matrix local to a node are referred to as the *probabilistic forwarding table* since they are used to randomly selecting the next hop in random walks.

III. Persistent Data Access through Decentralized Fountain Codes

We propose to use decentralized fountain codes to efficiently maintain data persistence in large-scale sensor networks. In this section, we first show how fountain codes can be used to access sensed data in a decentralized fashion, then present two data dissemination algorithms based on "one-way" random walks.

A. Decentralized Fountain Codes

We model a sensor network as a graph $G = (V, E)$, where V represents the set of sensors and E represents the set of links. Let N denote the total number of nodes in V. We assume that a subset of K nodes, called *sensing nodes*, monitor the environment and generate sensed data to be disseminated. The sensing nodes are able to cache their data on all other nodes (referred to as *caching nodes*), such that their data are available even when they fail or otherwise become unavailable.

In fountain codes, each encoded block is the XOR combination of a number of source blocks, randomly selected based on a special distribution, such as the Robust Soliton distribution in LT codes. To take advantage of fountain codes in sensor networks, one way to design a straightforward implementation is based on "two-way" random walks as follows. A node first generates the *code-degree* d from the Robust Soliton distribution, where the code-degree of a node refers to the degree of the encoded block cached on the node. This node may then request the source blocks from d distinct sensing nodes, uniformly distributed in the K sensing nodes, using random walks to deliver these requests. All d random-walk paths to the d sensing nodes are recorded in the network. Upon receiving the requests, the d sensing nodes disseminate their source blocks through the previous recorded d random-walk paths to the node. Finally, this node encodes the d received source blocks for later retrieval.

However, we believe that such "two-way" random-walk paths are rigid, not realistic and undesirable. *First*, sending data along the recorded random-walk paths makes the more demanding assumptions of bi-directional wireless links, which is sometimes not the case in reality. *Second*, to record random-walk paths in the network, sensors have to maintain forwarding tables with the size proportional to the number of random walks in the network. *Third*, sensor networks are inherently unreliable, and random-walk paths may be broken or lost if *any* of the sensors on these paths fails during the transmission. *Finally*, excessive overhead of transmitting control messages is unavoidable when selecting random sensing nodes, and additional random walks have to be activated if the selected node is not a sensing node, or a duplicate of previous selections.

In this paper, we seek to construct decentralized fountain codes with only one traversal of random walks, from sensing nodes to the caching nodes that encode and store the source blocks. Though nontrivial to design, such "one-way" random walks are much more scalable, completely stateless, and have avoided all the overhead and drawbacks of the aforementioned "two-way" walks.

Our algorithms are intuitively justified based on the following rationale. If the steady-state probabilities of two nodes are different in the random-walk Markov chain, *e.g.,* $\pi_i > \pi_j$, random walks guarantee that each source block has a higher probability to stop on node i than on node j. Therefore, node i will receive more distinct source blocks than node j, which results in node i obtaining a higher code-degree than node j.

Henceforth, we propose two heuristic algorithms to guarantee the Robust Soliton distribution of LT codes, in a completely decentralized fashion, and using "one-way" random walks.

These two algorithms are called Exact Decentralized Fountain Codes (EDFC) and Approximate Decentralized Fountain Codes (ADFC), since EDFC achieves the same coding performance of the original centralized LT codes, whereas ADFC offers degraded performance with the tradeoff of lower dissemination cost.

For both algorithms, we assume that, before execution, every node has been pre-programmed with the number of sensing nodes K, the total number of nodes N, and the Robust Soliton distribution. All these information can be broadcast to every node when the sensor network is initialized. However, K and N may decrease with time when some sensors die. In Section V, we will show that EDFC and ADFC do not require each node to know the precise value of K and N.

B. Exact Decentralized Fountain Codes (EDFC)

Because of the randomization introduced by random walks, the number of distinct source blocks received by a node is uncertain and usually does not equal the code-degree of this node. EDFC attempts to overcome such challenge and guarantee the Robust Soliton degree distribution. It is based on the observation that we may disseminate more than d source blocks on each node, but encode only d of them, where d is the code-degree of a node. Assume each node receives $x_d \cdot d$ source blocks on average, where x_d is called the *redundancy coefficient*. By choosing a sufficiently large x_d, the probability that such number of source blocks comprise less than d distinct source blocks can be made arbitrarily small. Therefore, the distribution of the number of source blocks used in encoding can be arbitrarily close to the Robust Soliton degree distribution. Note that, through the arguments of symmetry, x_d should depend only on d.

1) Algorithm Description: EDFC proceeds as follows. Initially, each node generates its code-degree from the Robust Soliton distribution. Next, each sensing node sends out its source block by random walks. As long as a source block stops at a node at the end of the random walk, this node will store this source block. After all source blocks are disseminated, each node generates its encoded block from a subset of received source blocks with cardinality equal to its code-degree.

Although the algorithm is simple, two critical elements need to be computed and are derived in details in the following: the number of random walks launched from each sensing node and the probabilistic forwarding tables for random walks.

The number of random walks: It is clear that the expected number of nodes with code-degree d in the network is $N\mu(d)$, where $\mu(d)$, the fraction of code-degree d nodes, is defined in (1). Each code-degree d node receives $x_d d$ source blocks. Therefore, the total number of source blocks received in the network is $\sum_{d=1}^{K} N\mu(d)x_d d$, which also equals bK, the number of source blocks disseminated from the K sensing nodes, where b denotes the number of random walks from each sensing node. Hence, we have

$$b = \frac{N\sum_{d=1}^{K} x_d d\mu(d)}{K}, \tag{3}$$

where x_d is solved later in (13) of this section.

Probabilistic forwarding tables: The probabilistic forwarding tables for random walks are computed by the Metropolis algorithm based on the required steady-state distribution of the random walks, which in turn is derived from the initially assigned Robust Soliton degree distribution. In the following, we present the derivation for the steady-state distribution in details.

Let π_d be the probability that a random walk stops at a given node with code-degree d. Since the total number of source blocks disseminated from all sensing nodes is bK, the total expected number of source blocks stopping at this node is $bK\pi_d$. At the same time, the expected number of source blocks stopping at this node also equals $x_d d$. Therefore, the stopping probability π_d can be computed by

$$\pi_d = \frac{x_d d}{bK}. \tag{4}$$

Substituting (3) into (4), we obtain the formula to compute the steady state distribution π_d for a node with code-degree d:

$$\pi_d = \frac{x_d d}{N\sum_{i=1}^{K} x_i i\mu(i)}. \tag{5}$$

To summarize, the steps of EDFC are as follows:

Step 1: ***Degree generation.*** Each node independently chooses its code-degree d from the Robust Soliton distribution.

Step 2: ***Compute steady-state distribution.*** Each node computes its steady-state distribution π_d from its chosen code-degree d based on the Robust Soliton distribution and the redundancy coefficients x_d by (5).

Step 3: ***Compute probabilistic forwarding table.*** Each node computes the probabilistic forwarding table by the Metropolis algorithm (2) from the steady-state distribution computed in Step 2.

Step 4: ***Compute the number of random walks.*** Each sensing node computes b, the number of random walks from itself, using (3).

Step 5: ***Block dissemination.*** Each sensing node disseminates b copies of its source block with its node ID by b random walks based on the probabilistic forwarding table computed in Step 3.

Step 6: ***Encoding.*** Each node generates an encoded block by bitwise exclusive-or (XOR) of a *subset* of d source blocks, uniformly distributed among all received source blocks. The d source node IDs of the source blocks are attached in the encoded block.

Note that if each node has more than one source blocks to be encoded in the network. Step 1 to Step 4 are required to be executed only once, whereas Step 5 and Step 6 are repeated for each source block.

The collector uses the original decoding algorithm of fountain codes to decode data. The source node IDs in each encoded block are used to construct the graph in the BP algorithm. Note that, since the average degree of an encoded block is $O(\ln(K/\delta))$ [3], the average number of source node IDs attached in each encoded block is also $O(\ln(K/\delta))$. Hence, this overhead can be ignored if the size of each block is much larger than the size of node IDs.

X:	the *chosen degree*, the initial assigned code-degrees on nodes in Step 1 of the algorithm
Y:	the *actual degree*, the actual number of distinct source blocks stopping at a node
Y_i:	the indicator variable: $Y_i = 1$ if at least one source block is received from the ith sensing node; $Y_i = 0$, otherwise.

TABLE I
THE DEFINITION OF RANDOM VARIABLES.

2) Overhead of EDFC: In EDFC, each node receives more source blocks on average than its code-degree, in order to achieve the Robust Soliton degree distribution. In this subsection, we estimate such overhead compared to the ideal algorithm, where the number of source blocks received at each node is exactly its code-degree.

Clearly, the transmission cost of EDFC and the ideal algorithm is the product of the number of random walks and the length of random walks, where the latter only depends on the network topology and is the same for both cases. Hence, the *overhead ratio* is defined as the ratio of the number of random walks. More precisely, in the ideal case, if no redundancy coefficient x_d is introduced, the number of random walks b_0 can be derived similarly as in (3):

$$b_0 = \frac{N \sum_{d=1}^{K} d\mu(d)}{K}. \tag{6}$$

Therefore, the overhead ratio is

$$g_1 = \frac{b}{b_0} = \frac{\sum_{d=1}^{K} x_d d\mu(d)}{\sum_{d=1}^{K} d\mu(d)} \ . \tag{7}$$

From the definition of the overhead ratio and intuition, the redundancy coefficient x_d governs the overhead, and hence should be minimized. In the following, we analyze, asymptotically, the required redundancy coefficients, and compute their minimal numerical values in practice, for EDFC.

Without loss of generality, we index the sensing nodes by $1, 2, \ldots, K$. We define random variables X, Y, and Y_i as shown in Table I. The efficient decoding of fountain codes requires that the degrees of the encoded blocks conform to the Robust Soliton distribution. To approximate such distribution, the probability $\Pr(Y < d|X = d)$, called *violation probability*, should be sufficiently small for all degrees less than or equal to the spike K/R as shown in Fig. 1.

Theorem 1: The violation probability $\Pr(Y < d|X = d)$ is $O(e^{-x_d d})$.

Proof: Clearly, $Y = \sum_{i=1}^{K} Y_i$. Since each source block has probability π_d to stop at a node with code-degree d, the probability that none of the b copies of the source block from the ith sensing node stops at a node with code-degree d is $\Pr(Y_i = 0|X = d) = (1 - \pi_d)^b$. Let $E = \sum_{i=1}^{K} x_i i \mu(i)$. We have

$$\begin{aligned} \Pr(Y_i = 1|X = d) &= 1 - (1 - \pi_d)^b \\ &= 1 - \left(1 - \frac{x_d d}{NE}\right)^{NE/K}, \end{aligned} \tag{8}$$

where the second equality is obtained by substituting π_d and b with (5) and (3). When $N \to \infty$, (8) becomes

$$\begin{aligned} \Pr(Y_i = 1|X = d) &\to 1 - e^{(-x_d d/\mathrm{E})(\mathrm{E}/K)} \\ &= 1 - e^{-x_d d/K} \ . \end{aligned} \tag{9}$$

Now, let p denote $\Pr(Y_i = 1|X = d)$. Given $X = d$, Y is a binomial random variable because Y_i are identical and independent Bernoulli random variables. Therefore, the violation probability is

$$\Pr(Y < d|X = d) = \sum_{j=0}^{d-1} \binom{K}{j} p^j (1-p)^{K-j}. \tag{10}$$

Equation (10) can be simplified as follows:

$$\begin{aligned} \Pr(Y < d|X = d) &\le \Pr(Y \le d|X = d) \\ &\le \binom{K}{d} (1-p)^{K-d} \\ &\le \left(\frac{eK}{d}\right)^d e^{-\frac{x_d d}{K}(K-d)} \\ &\simeq \left(\frac{K}{d}\right)^d e^{-x_d d}, \end{aligned} \tag{11}$$

where the first inequality is obvious, the second inequality is based on the tails of a binomial distribution [17], and the third inequality is due to the binomial bounds [17] and substituting p with (9).

■

Furthermore, since $\left(\frac{K}{d}\right)^d$ is an increasing function of d, its maximum is achieved when $d = K/R$. Hence,

$$\begin{aligned} \ln\left(\frac{K}{d}\right)^d &\le \frac{K}{R}\ln(R) \\ &= \sqrt{K}/(c\ln(K/\delta)) \cdot \ln(c\ln(K/\delta)\sqrt{K}). \end{aligned} \tag{12}$$

Equation (12) is $O(\sqrt{K})$, so $\left(\frac{K}{d}\right)^d$ grows slower than $e^{O(\sqrt{K})}$.

Theorem 1 provides a general rule for the tradeoff between coding performance and the communication overhead. Besides the asymptotic result of this theorem, we are further interested in practical values of the redundancy coefficient x_d. We next formulate an optimization problem to find the best x_d.

Our objective is to minimize the dissemination cost, which is governed by the number of random walks (3) from each sensing node as discussed in the definition of overhead ratio of this subsection. Therefore, the optimization objective is to minimize $\sum_{d=1}^{K} x_d d\mu(d)$, subject to the constraints that the violation probabilities should be sufficiently low for $1 \le d \le K/R$. This optimization problem is expressed as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{d=1}^{K} x_d d\mu(d) \\ \text{subject to} \quad & \Pr(Y < d|X = d) \le \delta_d \\ & x_d \ge 1 \\ & \text{for } d = 1, \ldots, K/R, \end{aligned} \tag{13}$$

where δ_d is a small constant and $\Pr(Y < d|X = d)$ is given in (10).

The above optimization problem can be solved off-line before network deployment. Therefore, its complexity is not a main concern. We may find a suitable local optimum by searching with reasonable initial values. As an example, in (13), we set the constraints of violation probabilities, δ_d, for

Redundancy Coefficient	x_1	x_2	x_3	x_4	x_5
Initial Search Point	3.0	2.3	2.0	2.0	1.5
Optimal Value	2.9956	2.3730	2.1005	1.9411	1.8341

TABLE II
OPTIMIZATION OF REDUNDANCY COEFFICIENTS.

$1 \le d \le K/R$, to 0.05, the number of nodes N and the number of sensing nodes K to 2000 and 1000, and the constants c and δ for the Robust Soliton distribution in (1) to 0.01 and 0.05, respectively. We then solve the resulting optimization problem by MATLAB, obtaining redundancy coefficients x_d as shown in Table II. Elements of the initial search point are also shown. Due to space constraint, we present data for only the first five code-degrees. Further numerical computation show that the overhead ratio g_1, as defined in (7), is 1.4508. This implies that, in EDFC, the additional transmission cost required in source-block dissemination can be less than half of the cost of the ideal algorithm.

C. Approximate Decentralized Fountain Codes (ADFC)

In this subsection, we present the second algorithm, ADFC, to approximate the Robust Soliton degree distribution in a distributed way. ADFC proceeds in a similar way as EDFC, but attempts to avoid its redundant random walks. ADFC is based on the following observation. In EDFC, although the initially assigned code-degree distribution on each node is the Robust Soliton distribution, the actual code-degree distribution is not exactly such distribution, due to the randomization introduced from random walks. Based on this, we design a new distribution $\nu(\cdot)$ to be a *hypothetical* chosen degree distribution such that the *actual* degree distribution are close to the Robust Soliton distribution.

1) Algorithm Description: The algorithm elements of ADFC, namely the number of random walks from each sensing node and the probabilistic forwarding table, can be derived in a similar way as in EDFC. We compute the number of random walks b by the same arguments used in (3). Let $\nu(\cdot)$ denote the chosen degree distribution. Then

$$b = \frac{N \sum_{d=1}^{K} d\nu(d)}{K}. \tag{14}$$

Furthermore, the steady-state distribution of the random walks is calculated similarly to (5):

$$\pi_d = \frac{d}{N \sum_{i=1}^{K} i\nu(i)}. \tag{15}$$

Next, we derive the chosen degree distribution $\nu(\cdot)$ initially assigned to nodes. We employ the same random variables as defined in Table I. Let $E = \sum_{i=1}^{K} i\nu(i)$. The conditional probability that at least one source block from the ith sensing node stops at a node, given the node's initially chosen code-degree is d is similar to (8):

$$\Pr(Y_i = 1 | X = d) = 1 - \left(1 - \frac{d}{NE}\right)^{NE/K}. \tag{16}$$

Let p denote $\Pr(Y_i = 1 | X = d)$. Since $Y = \sum_{i=1}^{K} Y_i$, we have the probability that a node with chosen code-degree d ends up having code-degree d' is

$$\Pr(Y = d' | X = d) = \binom{K}{d'} p^{d'} (1-p)^{K-d'}.$$

Thus, the actual degree distribution of a node is

$$\begin{aligned} \Pr(Y = d') &= \sum_{d=1}^{K} \Pr(X = d) \Pr(Y = d' | X = d) \\ &= \sum_{d=1}^{K} \nu(d) \binom{K}{d'} p^{d'} (1-p)^{K-d'} \end{aligned} \tag{17}$$

Let $\nu'(\cdot)$ denote the actual degree distribution $\Pr(Y = d')$. Then, to approximate the Robust Soliton distribution $\mu(\cdot)$, we minimize the mean-square error between $\nu'(\cdot)$ and $\mu(\cdot)$ for the degrees less than or equal to the spike K/R, as stated in the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{K/R} (\nu'(i) - \mu(i))^2 \\ \text{subject to} \quad & \sum_{i=1}^{K} \nu(i) = 1 \\ & \nu(i) \ge 0 \quad \text{for } i = 1, \ldots, K \ . \end{aligned} \tag{18}$$

To summarize, the steps of the ADFC are as follows:

Step 1: ***Degree generation.*** Each node independently chooses its code-degree d from the chosen degree distribution $\nu(\cdot)$, where $\nu(\cdot)$ is the result of the optimization problem (18).

Step 2: ***Compute steady-state distribution.*** Each node computes its steady-state distribution π_d from its code-degree d and the chosen degree distribution $\nu(\cdot)$ by (15).

Step 3: ***Compute probabilistic forwarding table.*** Each node computes the probabilistic forwarding table by the Metropolis algorithm (2) from the steady-state distribution computed in Step 2.

Step 4: ***Compute the number of random walks.*** Each sensing node computes b, the number of random walks from itself, using (14).

Step 5: ***Block dissemination.*** Each sensing node disseminates b copies of its source block with its node ID by b random walks based on the probabilistic forwarding table computed in Step 3.

Step 6: ***Encoding.*** Each node generates an encoded block by bitwise exclusive-or (XOR) of *all* received source blocks. The source node IDs of the source blocks are attached in the encoded block.

2) Overhead of ADFC: Similar to the overhead ratio of EDFC in (7), we define the overhead ratio of ADFC as

$$g_2 = \frac{b}{b_0} = \frac{\sum_{d=1}^{K} d\nu(d)}{\sum_{d=1}^{K} d\mu(d)}, \tag{19}$$

where b and b_0 is the number of random walks from each sensing node in ADFC and the ideal algorithm as defined in (14) and (6), respectively.

As an example, we present the numerical results from the optimization problem (18), where the total number of nodes N and the total number of sensing nodes K are set to 2000 and 1000, respectively. The additional parameters for

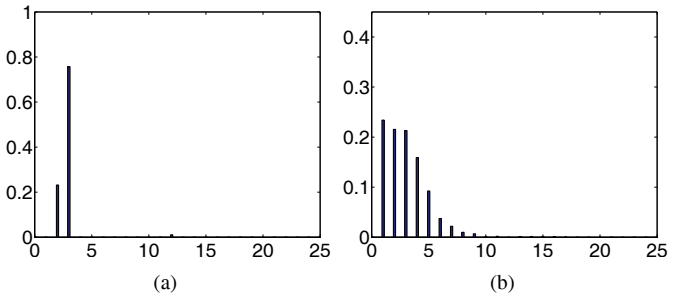

Fig. 2. (a) Chosen degree distribution. (b) Actual degree distribution. Parameters for the Robust Soliton distribution are $K = 1000$, $c = 0.01$, and $\delta = 0.05$, and the total number of nodes is 2000.

the Robust Soliton distribution are $c = 0.01$ and $\delta = 0.05$. The chosen degree distribution $\nu(\cdot)$ obtained from the optimization problem (18) and the actual degree distribution $\nu'(\cdot)$ are shown in Fig. 2. The actual degree distribution has a shape similar to the Robust Soliton distribution as shown in Fig. 1, though not exactly the same. Because of this inaccuracy, we expect that the collector needs to collect more encoded blocks than in EDFC to successfully decode. However, further numerical computation reveals that the overhead ratio g_2 is only 0.2326. This suggests that far less transmission cost is required to disseminate source blocks than EDFC or the ideal algorithm.

IV. Can Multiple Encoded Blocks do better?

In EDFC and ADFC, each node receives multiple distinct source blocks and encode them into one single encoded block. It may appear that using a single encoded block will lose some information in the received source blocks. If a node has ample storage space, does it improve the coding performance if different encoded blocks, from different subsets of the received source blocks, are maintained? The surprising answer is no.

Consider the extreme case where each node does not encode the received source blocks at all. Since all source blocks are kept, there is no information loss. Then, in the decoding process, the collector recovers the source blocks incrementally over a random subset of nodes. Theorem 2 shows that, even in this case, the collector needs to visit $\Omega(K)$ nodes to collect all K source blocks (among the collected blocks, many of them are duplicate). Therefore, no matter how the source blocks are encoded, $\Omega(K)$ surviving nodes are needed for all source blocks to be collected.

Theorem 2: When the code-degree distribution conforms to the Robust Soliton distribution, even if the source blocks on each node are not encoded, the collector must visit $\Omega(K)$ nodes in order to collect all source blocks with probability $1 - \delta$, where δ is a small positive number.

Proof: Let $Y_{i,j}$ be a random variable that assumes the value 1 if the source block j is collected when the collector visits the ith random node, and the value 0 otherwise. Let X_i be the number of source blocks on node i. We have

$$\begin{aligned}\Pr(Y_{i,j}=1) &= \sum_{d=1}^{K} \Pr(X_i = d)\Pr(Y_{i,j}=1|X_i=d) \\ &= \sum_{d=1}^{K} \mu(d)\frac{d}{K} \le \frac{c_1 \ln(K/\delta)}{K} \ , \end{aligned} \tag{20}$$

where c_1 is a constant, and the last inequality is a result from [3]: the average degree of an encoded block is $O(\ln(k/\delta))$.

Further define the random variable Z_j, such that Z_j has value 1 if the source block j is collected after the collector has visited M nodes, and has value 0 otherwise. We have

$$\begin{aligned}\Pr(Z_j = 0) &\approx \Pi_{i=1}^{M} \Pr(Y_{i,j} = 0) \\ &= \Pi_{i=1}^{M} (1 - \Pr(Y_{i,j} = 1)) \\ &\ge (1 - \frac{c_1 \ln(K/\delta)}{K})^M. \end{aligned} \tag{21}$$

The approximation above is because the random variables $Y_{i,j}$ are approximately independent when N is large.

Let E denote the event that all blocks are collected after the collector visits M nodes. Then

$$\begin{aligned}\Pr(E) &= \Pi_{j=1}^{K} \Pr(Z_j = 1) \\ &\le (1 - (1 - \frac{c_1 \ln(K/\delta)}{K})^M)^K \ . \end{aligned} \tag{22}$$

Hence, if we require that all blocks are collected with probability $1-\delta$ after the collector visits M nodes, *i.e.*, $\Pr(E) \ge 1-\delta$, we must have

$$(1 - (1 - \frac{c_1 \ln(K/\delta)}{K})^M)^K \ge 1 - \delta. \tag{23}$$

Applying logarithm to both sides of (23), we obtain

$$K \ln(1 - (1 - \frac{c_1 \ln(K/\delta)}{K})^M) \ge \ln(1-\delta) \simeq -\delta \ , \tag{24}$$

where the approximation of $\ln(1-\delta) \simeq -\delta$ is used since δ is small. Similarly, we obtain

$$-(1 - \frac{c_1 \ln(K/\delta)}{K})^M \ge -\delta/K \tag{25}$$

by using the fact that $\ln(1 - (1 - \frac{c_1 \ln(K/\delta)}{K})^M) \simeq -(1 - \frac{c_1 \ln(K/\delta)}{K})^M$ when M is large. By solving (25) using similar approximation, we obtain

$$M \ge K/c_1 \tag{26}$$

i.e., $M = \Omega(K)$. ■

V. Performance Evaluation

To evaluate the effectiveness and performance of the proposed algorithms, we have implemented both the original centralized and the decentralized implementations of fountain codes. The centralized implementation of fountain codes (LT codes in this case) consists of about 1000 lines of C++ code, with fully optimized implementation of the encoding and decoding algorithms. The decentralized implementation of fountain codes with random walks in wireless sensor networks is also simulated with C++.

We use the two-dimensional Geometric Random Graph [15], $G^2(N, r)$, as the topological model, where N sensors are uniformly distributed on a unit disc. Besides the total number of sensors N and the radio range r of each node, an additional parameter special to our algorithms is the total number of sensing nodes K. The K sensing nodes are uniformly distributed among the N sensors. We set $K = 10000$, $N = 20000$, and $r = 0.033$ in most experiments with exceptions explicitly stated. The average number of neighbors for each node is 21

in such a setting. To mitigate randomness, we show, for each data point in all figures, the average and the 95% confidence interval from 10 independent experiments.

A. Communication Cost and Decoding Ratio

We examine two main performance metrics, the *communication cost* and the *decoding ratio*. The communication cost, governed by the length of random walks and the number of random walks from a sensing node, represents the system efficiency. The decoding ratio denotes the number of nodes that need to be visited by a collector for decoding, normalized by the number of sensing nodes. It reflects the degree of fault tolerance of the network, since the fewer nodes are required for a collector to visit in order to decode all data, a higher percentage of nodes are allowed to fail.

We compare the performance of EDFC and ADFC with the two-way algorithm as described in Section III-A. First, the impact of random-walk lengths on the decoding ratio is studied. Fig. 3 plots the decoding ratio vs. the length of random walks for the three algorithms. In general, the decoding ratio decreases when the length of random walks increases, and stay stationary on a certain value if the length exceeds a threshold for all three algorithms. This is because the random walks approach the steady-state distribution when their lengths increase. In particular, for EDFC, Fig. 3 shows that when the random-walk length is larger than 500, the decoding ratio stays stationary around 1.05, which implies that EDFC achieves the same decoding performance of the original centralized fountain codes.

For ADFC, Fig. 3 shows that when the random-walk length is larger than 50, the decoding ratio stays around 1.6, higher than what is achievable by EDFC. This is because the actual degree distribution of ADFC is slightly different from the Robust Soliton distribution. However, ADFC requires much lower cost in data dissemination as we will show later.

For the two-way algorithm, the length of random walks shown in Fig. 3 is the one-way length. The decoding ratio of the two-way algorithm is less than EDFC and ADFC when the random-walk length is smaller than 500. This is because the uniform steady-state distribution in the two-way algorithm is easier to be achieved than the biased steady-state distribution of EDFC and ADFC [13]. Yet, this does not imply that the two-way algorithm has lower transmission cost than EDFC and ADFC for a given decoding ratio, since its additional random-walk overhead due to selecting a node that is not a sensing node, or a duplicate of previous selections, is not reflected in the length of random walks shown in the figure.

Next, we compare the communication costs of these three algorithms. For EDFC and the two-way algorithm, we record their minimal transmission costs when their decoding ratios are similar to centralized fountain codes. For ADFC, we record its minimal transmission cost when its decoding ratio is stabilized on some value (*e.g.,* 1.6 in Fig. 3). In addition, for the two-way algorithm, the transmission cost of control messages in selecting random nodes is considered the same as the cost in data transmission. Fig. 4 shows that the dissemination cost of ADFC is much lower than EDFC and the two-way algorithm. This is due to two reasons. First, the required length of random

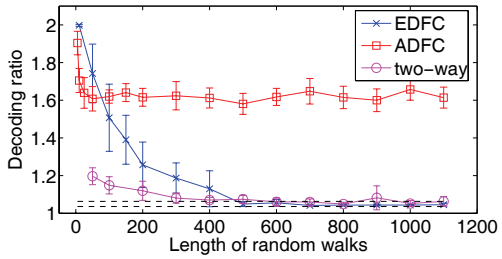

Fig. 3. The impact of the length of random walks on decoding ratio. The two dashed lines represent the 95% confidence interval for the decoding ratio of centraliz

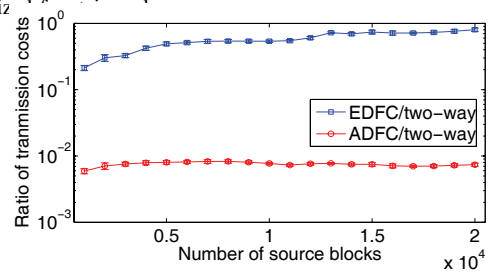

Fig. 4. The ratio of dissemination costs of EDFC and ADFC to that of the two-way algorithm.

walks of ADFC is shorter than both EDFC and the two-way algorithm as shown in Fig. 3. Second, for ADFC, the number of random walks from each sensing node is significantly smaller than that of EDFC and the two-way algorithm. For example, in the simulation setting of Fig. 3, we observe this number is 6 and 50 for ADFC and EDFC, respectively.

Furthermore, Fig. 4 shows that the cost ratio of EDFC to the two-way algorithm starts from 0.2 and increases slowly to 0.8, as the number of source blocks increases from 1000 to 20000. This cost ratio increases sub-linearly with the number of source blocks, since as shown in Theorem 1, the redundancy coefficient for EDFC grows sub-linearly with the number of source blocks. However, Fig. 4 shows that EDFC significantly outperforms the two-way algorithm for practical sensor networks with less than 20000 nodes. Furthermore, we emphasize that, as explained in Section III-A, EDFC is easier to implement and avoids the many disadvantages of the two-way algorithm.

B. Multiple Encoded Blocks Cannot Do Better

Theorem 2 shows that keeping multiple encoded blocks on each node does not offer any asymptotic performance advantage over keeping a single encoded block. Here, we further demonstrate that keeping multiple encoded blocks cannot provide significant benefits in practice. For this purpose, we simulate EDFC, except that the source blocks are not encoded on each node. Table III shows the number of nodes to be visited before the collector collects all source blocks, comparing simulation and numerical analysis results based on (26). We observe that the analytical lower bound K/c_1 is close to the actual mean number of required caching nodes in simulation. More importantly, the collector needs to visit close to K nodes even if the source blocks are not encoded.

C. Overestimation of K and N

In large-scale sensor networks, the failure of sensors are common events. If some sensors fail, the total number of

K	Sim. Mean	95% Conf. Interval	Analysis K/c_1
1000	938.80	[883.66, 993.94]	812.52
10000	7700.5	[6937.9, 8463.1]	7619.9

TABLE III
THE NUMBER OF NODES TO BE VISITED BEFORE THE COLLECTOR COLLECTS ALL SOURCE BLOCKS: SIMULATION VERSUS ANALYSIS

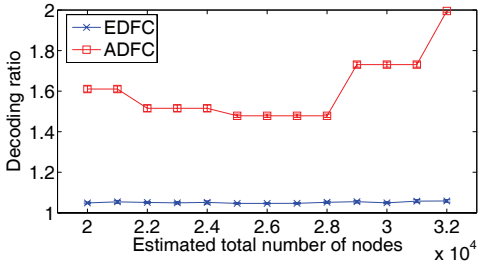

Fig. 5. The decoding ratio when N is overestimated. Actual $N = 20000$.

sensing nodes K and the total number of nodes N decreases. It is not feasible to update K and N to all nodes in the network whenever they change. A more reasonable protocol is to periodically update K and N with a sufficiently long time interval between two successive updates. In such a protocol, each node may overestimate K and N. In the following, we study the performance of EDFC and ADFC under such conditions.

First, we investigate the consequence of overestimating N. We fix the actual value of N to be 20000, while increasing the estimated N from 20000 to 32000. For EDFC, if N is overestimated, each sensing node disseminates more source blocks, which decreases the violation probability in (10) and helps decoding. This observation is validated by the experiment results shown in Fig. 5. Here we observe that the decoding ratio of EDFC stays close to 1.05 when the estimated N is larger than the actual value of N. For ADFC, the fraction of low-degree nodes decreases when more source blocks are disseminated to each node. Therefore, the performance of ADFC degrades, which is verified by the results shown in this figure, showing that the decoding ratio of ADFC increases when N is overestimated.

Next, we study the impact of overestimating K. We vary the actual K value from 1000 to 10000, while each node use a fixed estimated K value of 10000. Fig. 6 shows that the decoding ratio increases as the actual K value decreases, but successful decoding is achieved by both EDFC and ADFC. Furthermore, both figures suggest that EDFC is more robust than ADFC with overestimation of N or K.

VI. CONCLUSION

Wireless sensor networks may need to operate without sinks in remote geographical regions. In this paper, we seek to improve the fault tolerance and persistence of data in sensor networks by proposing a decentralized implementation of fountain codes, which efficiently disseminates original data throughout the network with random walks. We are attracted by the superior decoding performance and low decoding complexity of fountain codes as the number of nodes in the sensor network scales up, especially when compared to alternative coding techniques such as random linear codes. We have shown that asymptotically, as well as in actual experiments,

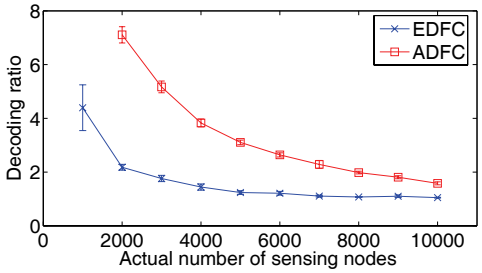

Fig. 6. The decoding ratio when K is overestimated. Estimated $K = 10000$.

the proposed algorithms are able to provide near-optimal fault tolerance with minimal demand on local storage.

REFERENCES

[1] B. Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," in *Proc. of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[2] A. Kamra, J. Feldman, V. Misra, and D. Rubenstein, "Growth Codes: Maximizing Sensor Network Data Persistence," in *Proc. of ACM SIGCOMM*, 2006.

[3] M. Luby, "LT Codes," in *Proc. of the 43th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

[4] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Distributed Fountain Codes for Networked Storage," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2006.

[5] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys (CSUR)*, vol. 26, no. 2, pp. 145–185, June 1994.

[6] Z. J. Hass and B. Liang, "Ad Hoc Mobility Management with Uniform Quorum Systems," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 228–240, April 1999.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage," in *Proc. of ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[8] J. S. Plank and M. G. Thomason, "A Practical Analysis of Low-Density Parity-Check Erasure Codes for Wide-Area Storage Applications," in *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, 2004.

[9] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2809–2816, June 2006.

[10] D. Wang, Q. Zhang, and J. Liu, "Partial Network Coding for Continuous Data Collection in Sensor Networks," in *Proc. of the Fourteenth IEEE International Workshop on Quality of Service (IWQoS)*, 2006.

[11] M. Rabbat, J. Haupt, A. Singh, and R. Nowak, "Decentralized Compression and Predistribution via Randomized Gossiping," in *Proc. of the Fifth International Symposium on Information Processing in Sensor Networks (IPSN)*, 2006.

[12] S. Acedanski, S. Deb, M. Medard, and R. Koetter, "How Good is Random Linear Coding Based How Good is Random Linear Coding Based Distributed Networked Storage?" in *First Workshop on Network Coding, Theory, and Applications (NetCod)*, 2005.

[13] S. Boyd, P. Diaconis, and L. Xiao, "Fastest Mixing Markov Chain on a Graph," *SIAM Review*, vol. 46, no. 4, pp. 667–689, December 2004.

[14] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Mixing Times for Random Walk on Geometric Random Graphs," in *Proc. of SIAM Workshop on Analytic Algorithmics & Combinatorics (ANALCO)*, 2005.

[15] P. Gupta and P. R. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, March 2000.

[16] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2001.