Energy Efficient Distributed Storage Systems with LT-Codes in Resource-Limited Wireless Systems

Louai Al-Awami
Computer Engineering Department
King Fahd University of Petroleum & Minerals (KFUPM)
Dhahran, Saudi Arabia
louai@kfupm.edu.sa

Hossam H. Hassanein
School of Computing
Queen's University
Kingston, ON, Canada
hossam@cs.queensu.ca

***Abstract*—In this paper we study the problem of designing a distributed data storage system using rateless codes for resource constrained systems such as Wireless Sensor Networks (WSNs). Rateless codes, e.g. LT-codes, can achieve reduced complexity of both encoding and decoding, which caters well to the nature of limited resources in such systems. However, data in WSNs is inherently decentralized and that poses an additional challenge when attempting to build codes with unconventional degree distributions. We propose an energy efficient distributed dissemination and coding scheme to build a decentralized LT-codes based storage over a network of resource-limited nodes to provide data survivability against possible failures. In the proposed scheme, each sensor node assigns selection probabilities to storage nodes using Robust Soliton Distribution (RSD) in a distributed fashion, and disseminates its data over the storage network randomly. The proposed scheme is compared to similar schemes in the literature by means of simulations. The results show that energy consumption can be substantially reduced while achieving the required storage requirements.**

***Keywords*—Data Survivability, Energy Efficiency, Wireless Sensor Networks, Fountain Codes, Decentralized Storage.**

I. Introduction

A new technological era is being witnessed and is enabled by advances in ubiquitous sensing, computing and connectivity. New paradigms such as Internet of Things (IoT), Big Data, Intelligent Transportation Systems (ITS), and Smart Homes, all benefit from the pervasiveness provided by WSNs and similar smart connected embedded systems. Despite their potential, WSNs are generally unreliable and suffer from limitation in energy, storage, and computing resources [1]. Therefore, it is crucial to equip WSNs with capabilities for data survivability as to strengthen their ability to cope with failures and data losses. This is all the more so when they are used for critical applications, or when network connectivity is intermittent such as the case in Delay Tolerant Networks (DTNs).

Distributed Data Storage Systems (DDSSs) can meet such requirements by implementing a distributed storage system that is resilient to data loss. DDSSs employ hardware redundancy and data replication to guarantee data survivability when failures occur. However, marrying resource-constrained wireless systems and DDSSs, is not so trivial, due to the limitation in storage and energy in the former. In addition, data in WSNs, for example, is inherently distributed, which calls for a distributed mechanism for building data storage. We believe that this can be achieved with the help of Fountain codes [2].

In this study, we propose a mechanism to construct a Fountain codes based DDSS from a set of physically decentralized sources in a distributed and energy efficient fashion. The concept of Fountain codes has been first introduced by Byers et al. in 1998 [2]. Later, M. Luby introduced Luby Transform (LT)-codes [3], the first realization of Fountain codes. Other codes have later been introduced including Online codes [4] and Raptor codes [5]. Despite the fact that Fountain codes were initially introduced for multicast applications, they have been applied to an ample range of scenarios in unicast and storage systems, among others. Fountain codes have also been adopted into standards such as DVB-IPTV for TV over IP service. For detailed information on Fountain codes and similar codes, readers can refer to [6].

Fountain codes enjoy a set of features that make them appealing to our settings. Given a set of k source packets, n encoded packets, the number of packets required for decoding to $(1+\epsilon)k$, where $\epsilon > 0$ is the decoding overhead, using a low complexity Belief Propagation (BP) decoder. Encoding on the other hand requires a simple bitwise xor circuit. The low complexity can be utilized to conserve energy which translates to longer network lifetime and lower operating costs. Further, simpler algorithms lower the overhead on the system resources allowing it to operate more efficiently. The main challenge however is that Fountain codes were meant for a centralized construction, and constructing them in a decentralized manner is not a trivial task.

In this paper we introduce a Decentralized Robust Soliton Storage (DRSS) which is an LT-codes-based DDSS that uses Robust Soliton Distribution (RSD). DRSS disseminates data in a random fashion by employing a selection probability policy derived from RSD. After nodes finish disseminating their data, they encode it and store it locally. The data can then be collected from nodes by a data collector in a random fashion as well. The proposed system enables a distributed implementation of an RSD based DDSS over a WSN where nodes are resource constrained and source data is decentralized. The goal of the scheme is to achieve data survivability while attaining both simplicity and energy conservation.

Our contributions include the introduction of a new data dissemination and encoding mechanism for building distributed data storage. We also show how RSD can be used to build a random selection policy enabling building a distributed storage using only local information. We evaluate the proposed scheme by simulation, and demonstrate how it can improve

978-1-4799-5952-5/15/$31.00 ©2015 IEEE

the performance of DDSS compared to existing schemes. Our evaluation addresses energy requirements, quality of resulting code, and robustness against nodes' failures.

The paper is organized as follows. Section II lays out some background and related work. Section III reviews LT-codes and their corresponding RSD degree distribution. The proposed scheme is discussed in Section IV and evaluation and comparison results are presented in Section V. Finally, Section VI concludes the paper.

II. Background and Related Work

In reference [8], A. Dimakis et al. describe Distributed Fountain (DF). DF implements an RSD-based distributed networked storage using a combination of a pulling mechanism and random walk with traps over a network of k source nodes and n storage nodes. The goal here is to allow the sensor data to be encoded over the set of storage nodes. In this fashion, a data collector will be able to recover the original k data packets using randomly selected encoded packets collected from any $(1+\epsilon)k$ storage nodes using a BP decoder, where ϵ is a small number. The authors assume that each node is aware of its location. They also assume the existence of a Greedy Perimeter Stateless Routing (GPSR) routing facility for delivering the data from the source nodes to the storage nodes. The scheme starts by each storage node sampling a random number d from a degree distribution $\rho(i)$. Then, the storage node contacts d randomly chosen nodes with the hope of reaching d source nodes. If the target node happens to be a source node, it replies back with its data packet to the storage node who initiated the request. If, on the other hand, the target node happens to be another storage node, then the target node starts a random walk in search of a nearby source node. The random walk terminates at the first source node it visits, and data from that source node is sent to the requesting storage node. Using results on random walks with traps, the authors show that the random walk part of the process diminishes asymptotically for large n, which means the algorithm has a complexity of $O(\sqrt{n})$. The main disadvantage of this scheme is that communication is initiated by storage nodes which can substantially impact the energy required due to the two-way trip.

In reference [9], the authors propose the implementation of an LT-codes based distributed network storage for the purpose of data persistence. The proposed schemes employ random walk to disseminate data from source nodes over a network of storage nodes. The work includes two algorithms, Exact Decentralized Fountain Codes (EDFC) and Approximate Decentralized Fountain Codes (ADFC). While EDFC strives to achieve the exact distribution of the original centralized LT-codes, the ADFC achieves a degraded distribution at a lower cost of implementing the code. Both algorithms are comprised of the following steps:

1) Each node chooses its degree d from the designated degree distribution.
2) Using d and a redundancy coefficient x_d, each node calculates π_d, the steady-state distribution.
3) Using the Metropolis algorithm [10], each node computes its probabilistic forwarding table.
4) Each source node computes the number of required random walks b.
5) Each source node disseminates b copies of its source data using the probabilistic forwarding table.
6) Each node chooses d of the source packets it receives and combines them using bitwise *xor*. The source IDs are attached to the encoded packets to identify the sources of the combined packets.

In step (3), starting with the steady-state distribution $\pi = \{\pi_1, \pi_2, ...\}$, the Metropolis algorithm generates the transition matrix $P = [P_{ij}]$ as follows:

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

where $N(i)$ denotes the set of direct neighbors of node i and D_m is the maximal node degree in the network (graph).

The two algorithms differ in that EDFC strives to achieve the exact RSD, ADFC targets an approximation of the RSD to eliminate some redundancy from the original distribution to reduce the number of steps required for encoding. In this study, we only cover EDFC since we are interested in exact distribution which is expected to lead to a better performance when matched with BP decoder for data recovery applications. Note that both algorithms require synchronization between nodes to know when the random walks terminate for the encoding to take place. Besides, both algorithms require a neighbour discovery mechanism to build the forwarding table, which is essential for the algorithms to work.

III. LT Codes and Robust Soliton Distribution

In this section, we review the operation of LT-codes and it's degree distribution, i.e. RSD. This background is essential to the discussion on the development of the proposed scheme. To encode a set of k *source packets*, the decoder of an LT-codes samples the degree distribution $\rho(i)$ for a value d, where $1 \geq d \geq k$. Next, d packets are selected uniformly at random from the k source packets. The chosen packets are then combined through a bitwise *xor* to generate an *encoded packet* which is sent to the destination. d is referred to as the *packet degree*. Let $X_i = \{x_1, x_2, ..., x_d\}$ be the set of source packets chosen by the encoder at step i, then an encoded packet e_i can be generated as $e_i = x_{i_1} \oplus x_{i_2} \oplus ...x_{i_d} \forall x_{i_j} \in X_i$..

On the other hand, the BP decoder at the destination works iteratively. After receiving enough packets, a bipartite graph is generated where each encoded packet is connected to its source packets. Then, at each step, the decoder searches for encoded packets with degree one, called *ripple*. Since the value(s) of the packets in the ripple can be determined (decoded), the edges connected to such packets can then be removed. Further, the value(s) of the source packets connected to the decoded packets at the current step are modified to reflect the change in the graph. This later step reduces the degree of some encoded packets from degree two to degree one, called *release*, causing the next iteration to start. The decoder repeats this process until all encoded packets are decoded.

Note that when the decoder arrives at a stage where no degree one packets are available, the decoder fails. In such case, the decoder either signals a failure, or waits for more packets to arrive in hope of resuming the decoding process.

BP decoder is attractive because of its simple and low complexity decoding. In comparison to the Gaussian Elimination (GE) with a complexity tag of n^3, BP reduces the decoding complexity to $n\log(n)$.

The degree distribution, $\rho(i)$, used by LT-codes is RSD. The distribution can be described by first introducing the probability mass function (pmf) of the Ideal Soliton Distribution (ISD):

$$\rho(i) = \begin{cases} \frac{1}{k}, & \text{for } i = 1 \\ \frac{1}{i(i-1)} & \text{for } i = 2, 3, ..., k \end{cases} \quad (2)$$

The idea behind ISD is to increase the probability of low degree packets to guarantee that the ripple is never empty and therefore decoding never stalls. Despite working in theory, ISD performs poorly in practice. The reason is that any deviation in the expected behaviour causes the decoder to stale when no packets of degree-one exist. Therefore, Luby in reference [3] proposed the RSD to be used as the degree probability distribution. RSD is defined by the following pmf

$$u(i) = \frac{\rho(i) + \tau(i)}{Z} \quad (3)$$

where

$$Z = \sum_i \rho(i) + \tau(i) \quad (4)$$

and

$$\tau(i) = \begin{cases} \frac{S}{i \times k}, & \text{for } i = 1, 2, ..., \frac{k}{S} - 1 \\ \frac{S}{k} \ln(\frac{S}{\delta}) & \text{for } i = \frac{k}{S} \\ 0 & \text{for } i > \frac{k}{S} \end{cases} \quad (5)$$

where

$$S = c \ln\left(\frac{k}{\delta}\right)\sqrt{k}. \quad (6)$$

δ is the probability of failure and c is a constant of order one. Typically, $c = 0.2$ have been shown to work well.

IV. Decentralized Robust Soliton Storage

This section presents the proposed Decentralized Robust Soliton Storage (DRSS). The scheme assumes a network $N(k, n)$ comprising a set of k *source nodes*, $U = \{u_1, u_2, ..., u_k\}$. As with similar distributed storage schemes [8] [9], we add a set of n redundant *storage nodes* to achieve additional reliability. We denote the set of storage nodes by $V = \{v_1, v_2, ..., v_n\}$. Storage nodes are assumed to be able to *store* and *relay* data. Source nodes can, in addition to storing and relying, generate data through sensing. Therefore, we will use "storage nodes" to refer to all the nodes in the network. So, the new network size, n, is defined as $n = (s + 1)k$, for some value $s > 0$. We refer to s as *data survivability* [11].

Storage nodes are assumed to have enough space to store packets received from source nodes and to perform encoding. However, after encoding, only encoded packets and encoding vectors are stored. We do not address communication errors, so all packets are assumed to be received correctly.

The scheme consists of the following four main steps:

1) **Packet Degree Assignment** (Q_i): The first step is to assign a degree $d(i)$ to each storage node. $d(i)$ indicates the desired packet degree of the encoded packet that node v_i is to store. This assignment is performed in a decentralized way, since the mapping between each storage node and its degree can be computed at each source node independently using only knowledge of k and RSD. The mapping is represented by $Q = \{Q_1, Q_2, ..., Q_k\}$, where Q_i refers to the set of nodes that will be storing packets of degree i. Each storage node, on the other hand, only calculates its own $d(i)$, which is to be used when encoding source packets that are received.
2) **Selection Probability Computation** ($P(i)$): Each source node u_i calculates the selection probability of each storage node based on the packet degree assigned to the storage node in the previous step. We denote by $P(i)$ the probability of selecting a node with packet degree i.
3) **Packet Forwarding**: Source nodes disseminate source data packets using the selection probability generated in (2). At each step, the source node:
   - Selects a storage node v_j.
   - Forwards data packet to v_j.
   - Removes v_j from storage nodes selection poll.
   - Modifies $P(i)$ accordingly.

   This process is repeated m times, where m is called the Redundancy Factor (RF), and it is the required number of data packets to be forwarded. The modification of $P(i)$ that takes part at every step as well as the required RF will both be discussed later.
4) **Encoding**: Each storage node v_i chooses randomly $d(i)$ packets from the packets it received and encodes them using *bitwise xor*. In addition to the encoded packet, each storage node stores a 1-dimensional binary vector of size k with each entry equals to 1 if the packet from u_i has been used to generate the encoded packet at this node, and 0 otherwise.

Assume storage nodes are numbered from 1 to n. We are interested in partitioning the storage nodes into groups pertaining to different packet degrees between 1 and k, where nodes in group Q_i stores packets of degree i. Then,

$$Q_i = [v_{(q_{(i-1)}+1)}, ..., v_{(q_i)}]$$

$$q_i = \begin{cases} 0, & \text{for } i = 0 \\ n \sum_{j=1}^{i} u(j) & \text{for } i = 1, 2, ..., n \end{cases} \quad (7)$$

Now, given an RSD as defined by Eq.3, and substituting the values of $\rho(i)$ and $\tau(i)$ then,

$$u(i) = \begin{cases} \frac{1}{Z}[\frac{1}{k} + \frac{S}{k}], & \text{for } i = 1 \\ \frac{1}{Z}[\frac{1}{i(i-1)} + \frac{S}{ik}] & \text{for } 1 < i < \frac{k}{S} \\ \frac{1}{Z}[\frac{1}{i(i-1)} + \frac{S}{k}\ln(\frac{S}{\delta})] & \text{for } i = \frac{k}{S} \\ \frac{1}{Z}[\frac{1}{i(i-1)}] & \text{for } \frac{k}{S} < i \leq n \end{cases} \quad (8)$$

The Commulative Distribution Function (CDF) of $u(i)$ can be expressed for $i \geq \frac{k}{s}$ as

$$\begin{aligned} P(i) = \sum_{j=1}^{i} u(i) =& \frac{1}{Z}[\frac{1}{k} + \frac{S}{k} + \frac{1}{2} + \frac{S}{ik} + \frac{1}{6} + \frac{S}{ik} + ... + \frac{1}{i(i-1)} \\ &+ \frac{S}{k}\ln(\frac{S}{\delta}) + \frac{1}{i(i+1)} + \frac{1}{(i+2)(i+1)} + ...] \\ =& \frac{1}{Z}[\frac{1}{k} + \frac{S}{k}(\ln(\frac{S}{\delta}) + \sum_{i=1}^{\frac{k}{S}-1}\frac{1}{i}) + \frac{i}{i+1}] \end{aligned}$$

Therefor, q_i can be expressed as

$$q_i = \begin{cases} \frac{n}{Z}[\frac{1}{k} + \frac{S}{k}] & \text{for } i = 1 \\ \frac{n}{Z}[\frac{1}{k} + \frac{S}{k}(\sum_{i=1}^{\frac{k}{S}-1} \frac{1}{i}) + \frac{\frac{k}{S}-2}{\frac{k}{S}-1}] & \text{for } 2 \leq i \leq \frac{k}{S} - 1 \\ \frac{n}{Z}[\frac{1}{k} + \frac{S}{k}(\ln(\frac{S}{\sigma}) + \sum_{i=2}^{\frac{k}{S}-1} \frac{1}{i}) + 1] & \text{for } i = \frac{k}{S} \\ \frac{n}{Z}[\frac{1}{k} + \frac{S}{k}(\ln(\frac{S}{\sigma}) + \sum_{i=1}^{\frac{k}{S}-1} \frac{1}{i}) + \frac{i}{i+1}] & \text{for } \frac{k}{S} + 1 \leq i \leq k \end{cases}$$

In our scheme, RSD is used to determine the degree of each storage node. The other component we need to determine is the probability of selection of each node. By weighing nodes differently, we can achieve the required distribution. The basic idea is to have each node of degree i with probability p_i proportional to its degree. More formally, let p_1 be the probability of nodes with degree 1. Then

$$p_1 = (\sum_{i=1}^{n} d(i))^{-1},$$

where $d(i)$ is the degree of node i. The probabilities of the remaining degrees are computed using the following relation

$$p_i = d(i) \times p_1 \tag{9}$$

Another important parameter is the RF which is the number of packets sent out by each source node (m). Remember that the total number of packets required is equal to $\sum_{i=1}^{n} d(i)$. Accordingly, RF can be computed as:

$$m = (\sum_{i=1}^{n} d(i))/k$$

An important difference between this scheme and other schemes, is that the selection policy is modified after each node is selected. Whenever a node is selected, its removed from the poll of possible future nodes. The selection probabilities of the remaining nodes are then normalized to reflect this change while maintaining their relative weight.

Let v_i be the storage node selected at the last step. The selection probability, at the current source node, of every remaining node v_j is computed as

$$P(v_j) = \frac{P(v_j)}{\sum_{l} P(l) \ \forall l \neq i}$$

The process is repeated until m packets have been disseminated. After this process finishes, each storage node v_i chooses randomly $d(i)$ packets form its buffer and encodes them linearly. If the number of packets in the buffer is less than $d(i)$, all the packets are encoded. To recover the source packets from the storage nodes, we need to contact random storage nodes to collect encoded packets while progressively decoding using a BP decoder. The decoding process continues until all k source packets are decoded. Algorithms 1 and 2 summaries the steps at the source and storage nodes, respectively.

Algorithm 1 DRSS (Source Node)

Input: k,s, and x_i (*sensor data*)
  $n \leftarrow k \times (s + 1)$
  Generate node degree mapping Q and $d(i)$ for all nodes
  Build selection probability $P(i)$
  $m \leftarrow \frac{\sum_{i=1}^{n} d(i)}{k}$
  for $j = 1 \rightarrow m$ **do**
    Selects target storage node v_j as in $P(i)$
    Forward data x_i to v_j
    $P(i) = \frac{P(v_j)}{\sum_{l} P(l) \ \forall l \neq i}$
  end for

Algorithm 2 DRSS (Storage Node)

Input: k and s
  $n \leftarrow k \times (s + 1)$
  Generate node degree mapping Q and $d(i)$ for this node
  Receive data packets (x_i)'s and store in X_j
  Select $d(j)$ packets uniformly at random
  Encode data as $e_j = x_{j_1} \oplus x_{j_2} \oplus ...x_{j_d} \forall x_{j_i} \in X_j$

V. Performance Evaluation

The evaluation of the proposed scheme has been carried out using simulations. We have also implemented the two schemes proposed in references [8] and [9] for comparison. Other similar schemes exist in the literature, but they target other distribution such as Raptor codes, e.g. [12], [13], and [14]. We are mainly interested in the collective performance of each scheme in three areas: quality, efficiency, and robustness.

*1) **Quality**:* By quality we mean how close the resulting degree distribution is to the RSD. To measure this quantitatively, we use the Hellinger Distance f-divergence measure [15]. There exist other measures of divergence between probability distributions such as KL-Divergence [16]. One limitation of KL-Divergence compared to Hellinger Distance is that it requires - by definition - all probabilities of the possible outcomes to be > 0. In addition, while Hellinger Distance is dimensionless, KL-Divergence is not. This makes Hellinger Distance easier to interpret and more applicable to our problem since zero probability of certain code degrees is possible.

The Hellinger Distance $H(P, Q)$ between two probability distributions $P = \{p_1, p_2, ..., p_k\}$ and $Q = \{q_1, q_2, ..., q_k\}$ quantifies the similarity between P and Q. It is defined as

$$H(P, Q) = \frac{1}{\sqrt{2}} \sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2 \tag{10}$$

Basically, $H(P, Q)$ returns a value in the interval $[0, 1]$ where 0 indicates that P and Q are exactly the same, i.e. $p_i = q_i \forall i$, while 1 implies that $p_i \neq q_i \forall i$. In our case, P is the target RSD, while Q is the degree distribution of the code generated by the scheme under consideration.

*2) **Efficiency**:* Efficiency refers to the energy efficiency. It is obvious that the energy consumed is proportional to the number of hops traveled by data packets. Therefore, we use the total number of hops as an indicator to the energy in our cost equation as will be discussed shortly.

TABLE I. HELLINGER DISTANCE (H) FOR DRSS, EDFC, AND DF.

k	n	Hellinger Distance		
		DRSS	EDFC	DF
50	169	0.114	0.2512	0.190
75	256	0.105	0.234	0.168
100	344	0.102	0.236	0.171
125	400	0.096	0.239	0.173
150	484	0.092	0.230	0.161

We assume a network composed of wireless nodes powered by batteries. The energy consumed by nodes can be due to either sensing (in the case of source nodes), communications (transmitting/receiving), or encoding. The corresponding energies consumed are therefore represented by ξ_s, ξ_t,ξ_r, and ξ_e, respectively, where $\xi_t, \xi_r >> \xi_e$. The energy of the data collector is assumed to be infinite.

*3) **Robustness**:* Robustness refers to the reliability of the code against failures. One indicator of interest is the ratio of the nodes unreachable by the dissemination protocol and thus has no packets. We refer to these as "void packets". The intuition behind void packets is that more packets means more redundancy and hence more robustness.

The simulator works on a network of $n = (s+1)k$ nodes, where only k of them are source nodes. For every iteration in the simulation, the following steps are executed:

1) Generate a network of $n = (s+1)k$ nodes based on the supplied values of k and s, and initialize them.
2) Every source node disseminates its data according to the designated scheme.
3) Storage nodes encode/relay data packets.
4) Calculate energy used (Ξ), degree distribution, Hellinger Distance (H), and the number of void packets (β_0).
5) Calculate probability of successful decoding, and Decoding Overhead (DO) (β), for different failure levels. (P_s)

These steps are repeated for every scheme of the three and for different values of k. All schemes use the same instance of RSD with parameters of $\delta = 0.05$ and $c = 0.2$.

First, we evaluate the Hellinger Distance (H) to compare how close the resulting degree distribution to the target RSD. As seen in Table I, DRSS results in a smaller value of H and thus better approximation of RSD. However, DF outperforms both DRSS and EDFC in the resulting degree distribution. This is expected since DF allows the storage nodes to construct packets with the exact degree generated by RSD. Of course, this comes at a cost of more energy resulting from the two-way communication needed to acquire data packets.

The second metric of interest is the energy required to implement the coding schemes. In this study, we calculate the energy needed for every scheme based on the CC1000 chip. The power requirements of this chip are 5, 25, and 28.8 mW, for processing, sending, and receiving, respectively. Table II shows the number of operation required by each scheme for different network sizes. The operations of interest that contribute to the energy are encoding, sending, and reception. Since we are assuming no transmission errors, the number of send and receive operations are equal for the same scheme. We also define (Ξ) to the total energy required for code construction including data dissemination and encoding. Ξ will be used later to compare the schemes under consideration.

TABLE II. POWER REQUIRED TO BUILD RSD STORAGE.

k	n	Operation	DRSS	EDFC	DF
50	169	COD (ζ_e)	683	582	678
		SND (ζ_t)	3967	15019	10885
		REC (ζ_r)	3967	15019	10885
Total Power (Ξ)[*in watts*]			215.65	810.93	585.74
75	256	COD (ζ_e)	1068	923	1084
		SND (ζ_t)	7887	29472	20946
		REC (ζ_r)	7887	29472	20946
Total Power (Ξ)[*in watts*]			427.29	1590.2	1126.03
100	344	COD (ζ_e)	1485	1359	1550
		SND (ζ_t)	11750	48541	11750
		REC (ζ_r)	11750	48541	11750
Total Power (Ξ)[*in watts*]			636.05	2618.3	1727.99
125	400	COD (ζ_e)	2089	1838	2106
		SND (ζ_t)	18626	65217	47401
		REC (ζ_r)	18626	65217	47401
Total Power (Ξ)[*in watts*]			1006.94	3517	2546.48
150	484	COD (ζ_e)	2492	2248	2612
		SND (ζ_t)	24642	118328	63528
		REC (ζ_r)	24642	118328	63528
Total Power (Ξ)[*in watts*]			1330.81	6377.29	3411.81

The third metric that is important to consider is the ratio of encoded packets with degree zero. We refer to these as *void packets* and we denote by β_0 the ratio of void packets to the total number of source nodes. Let x_0 be the total number of void packets in a code over a network of n storage nodes. Then β_0 can be computed as $\beta_0 = x_0/k$.

β_0 quantifies the fraction of storage nodes not reached by the dissemination mechanism. Table III shows β_0 for the three schemes. In the case of DF, $\beta_0 = 0$ because the algorithm starts from storage nodes. What is noticeable is that EDFC suffers from a high average β_0 value. This is mainly due to the random walk mechanism. This could imply that the availability of the data at different parts of the network is not uniform.

TABLE III. VOID PACKETS RATIO (β_0) FOR DRSS, EDFC, AND DF.

k	n	No. void packets (β_0)		
		DRSS	EDFC	DF
50	169	0.5(0)	2.06(0.041)	0(0)
75	256	0.04(0)	2.44(0.032)	0(0)
100	344	0.09(0)	2.24(0.022)	0(0)
125	400	0.01(0)	2.24(0.018)	0(0)
150	484	0.03(0)	3.06(0.020)	0(0)

To see the impact of void packets on the robustness of the code, consider the probability of successful decoding (P_s). Fig. 1 shows that as the number of nodes failing increases, DRSS maintains a medium P_s indicating that code survivability is between the other two. The explanation of this observation is different for EDFC and DF. Fountain codes are not meant to necessarily increase survivability of the data but to lower decoding complexity. This is why, the closer the code to the RSD, the lower the decoding complexity, but not necessarily the higher survivability it has. Moreover, EDFC's random walk mechanism results in more localized dissemination of data, i.e. a random walk does not distribute the data over the network uniformly. Hence, when failure occurs in certain region of the network, the code may not be decodable.

We finally, present the Divergence-Effeciency-Robustness Indicator (DER) (Ω) to be a cost function to compare the

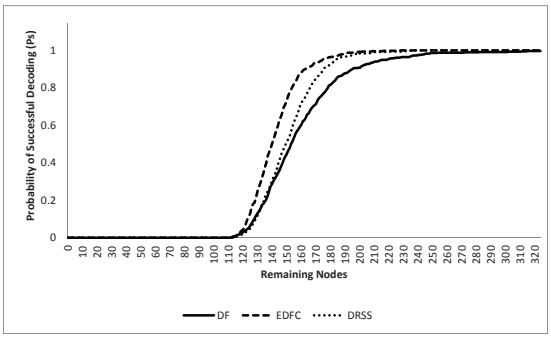

Fig. 1. Probability of Successful Decoding (P_s): $k = 100$, $s = 3$, $\mathbb{F}_2$.

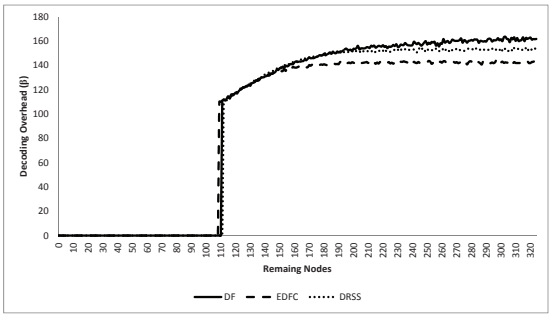

Fig. 2. Decoding Overhead (β): $k = 100$, $s = 3$, $\mathbb{F}_2$.

three schemes by capturing all the three aspects at once. DER is defined as

$$\Omega = \Xi(1 + \beta_0 + H) \tag{11}$$

The idea behind the metric is to scale the energy proportionally to the ratio of void packets, as well as the divergence from the required degree distribution. The scaling represents a penalty for the inferior performance.

TABLE IV. DER (Ω) FOR DRSS, EDFC, AND DF.

k	n	DER		
		DRSS	EDFC	DF
50	169	348	819	697
75	256	489.4	2013	1315
100	344	758	3294	2023
125	400	1113	3664	2986
150	484	1492	7972	3960

Table IV shows the different values of the DER indicator. As it is clear from the table, DRSS achieves a better overall efficiency for different values of k. The main point to be made, is that the saving increases with the network size. This implies that not only can DRSS save energy, but also that it provides better scalability. Whereas DF requires about twice the energy needed by DRSS, EDFC requires situationally more, due to the convergence characteristics of the random walk.

VI. CONCLUSION

Fountain codes are attractive because of their low-complexity encoding and decoding. However, when the source data is decentralized, building Fountain codes becomes challenging. We proposed DRSS, a scheme that aims at implementing LT-codes based distributed storage in an energy efficient manner. We compare the new scheme to two similar schemes in the literature; DF and EDFC, and show that DRSS achieves the required degree distribution with a fraction of the energy required by the other schemes even in small settings. We strongly believe that this can prove the potential and the suitability of LT-codes to provide data survivability in future application of WSNs and enable more reliable operations.

ACKNOWLEDGMENT

The authors would like to thank Dr. Shahram Yousefi for the fruitful comments and discussions. This research is supported by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

[1] K. Xu, H. Hassanein, G. Takahara, and Q. Wang, "Relay Node Deployment Strategies in Heterogeneous Wireless Sensor Networks," *Mobile Computing, IEEE Transactions on*, vol. 9, no. 2, pp. 145–159, Feb 2010.

[2] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," in *ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. New York, NY, USA: ACM, 1998, pp. 56–67.

[3] M. Luby, "LT Codes," in *43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 271–280.

[4] P. Maymounkov, "Online Codes," New York Univeristy, Technical Report TR2002-833, 2002.

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

[6] K. Rao, *Channel Coding Techniques for Wireless Communications*. New Delhi: Springer, 2015.

[7] M. Albano and S. Chessa, "Replication vs Erasure Coding in Data Centric Storage for Wireless Sensor Networks," *Computer Networks*, vol. 77, pp. 42 – 55, 2015.

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

[9] Y. Lin, B. Liang, and B. Li, "Data Persistence in Large-Scale Sensor Networks with Decentralized Fountain Codes," in *26th IEEE International Conference on Computer Communications (INFOCOM)*, 2007, pp. 1658–1666.

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

[11] L. Al-Awami and H. Hassanein, "Data Survivability for WSNs via Decentralized Erasure Codes," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, Aug 2012, pp. 94–99.

[12] S. Aly, K. Zhenning, and E. Soljanin, "Raptor Codes based Distributed Storage Algorithms for Wireless Sensor Networks," in *IEEE International Symposium on Information Theory (ISIT 2008)*, July 2008, pp. 2051–2055.

[13] Z. Kong, S. A. Aly, and E. Soljanin, "Decentralized Coding Algorithms for Distributed Storage in Wireless Sensor Networks," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 261–267, 2010.

[14] C. Stefanovic, V. Stankovic, M. Stojakovic, and D. Vukobratovic, "Raptor Packets: A Packet-Centric Approach to Distributed Raptor Code Design," in *IEEE International Symposium on Information Theory (ISIT 2009)*, June 28-July 3 2009, pp. 2336 –2340.

[15] E. Hellinger, "Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen." *Fr Die Reine Und Angewandte Mathematik*, 1909.

[16] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951.