Computer Networks 128 (2017) 41–50

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Robust decentralized data storage and retrieval for wireless networks

Louai Al-Awami[a,*], Hossam S. Hassanein[b]

[a] Department of Computer Engineering, King Fahd University of Petroleum & Minerals, Dhahran, KSA
[b] School of Computing, Queen's University, Kingston, ON, Canada

ARTICLE INFO

Article history:
Received 5 July 2016
Revised 5 December 2016
Accepted 9 February 2017
Available online 17 February 2017

Keywords:
Wireless sensor networks
Decentralized storage
Data survivability
Energy efficiency
Fountain codes

ABSTRACT

In this paper we study the problem of distributed data storage using Rateless codes for large-scale resource-constrained wireless networks. We focus on building a robust storage system using Fountain codes from physically decentralized sources. Fountain codes, e.g. LT-codes, can achieve reduced complexity of both encoding and decoding, which caters well to the nature of such networks. 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 is assigned a selection probability derived from a Robust Soliton Distribution (RSD) in a distributed fashion. Source nodes then disseminate their data over the storage network randomly, in accordance with the selection probabilities. The proposed scheme is compared to similar schemes in the literature by means of simulations. We evaluate the energy required for building the storage as well as the energy needed for data retrieval from the storage system. Results show that energy consumption can be substantially reduced while achieving the required storage requirements.

1. Introduction

We are witnessing a new technological era that is enabled by advances in ubiquitous sensing, computing and connectivity. New applications such as Internet of Things (IoT), Intelligent Transportation Systems (ITS), smart grid, and smart homes, all benefit from the pervasiveness provided by connected smart embedded devices. Despite this potential, there seems to be a constant need for alternative paradigms to cope with the challenges and requirements that come about with these applications. For instance, while IoT devices generate immense amounts of data, their underling system, consisting of tiny devices, is generally unreliable and suffers from limitations in energy, storage, and computing resources [1]. It is therefore crucial to search for unconventional alternatives to strengthen their ability to cope with failures and data losses while respecting their limitations. This is all the more so when such systems are used for critical applications, or when network connectivity is intermittent such as the case with Delay Tolerant Networks (DTNs).

Recently, a number of proposals have emerged on the use of Distributed Data Storage Systems (DDSSs) to provide data availability and increase data survivability against failures [2]. DDSSs can facilitate such requirements by implementing a distributed storage system that is resilient to data loss by employing hardware redundancy and data replication to guarantee data survivability when failures occur. The use of redundancy enables data to be accessed from multiple locations, and therefore increases availability. However, using DDSS in resource-constrained wireless systems is not trivial, due to the limitations in storage, computing, and energy.

This has led to suggesting the use of Fountain codes [3] to construct DDSS from a set of physically decentralized sources in a distributed and energy efficient fashion [4–6]. DDSSs can benefit greatly from the versatility of Fountain codes in numerous ways. Consider, for example, the following application. A Wireless Sensor Network (WSN) becomes fragmented and sensor nodes lose connectivity to the sink node. The network runs a self-healing mechanism to restore network connectivity. Such a restoration process may not be momentary and requires time. To protect data, sensor nodes can resort to Fountain codes, to increase data persistence, by exchanging their data to build a storage using LT-codes. After the network connectivity is restored, data can be decoded by sensor nodes or a sink node. Since encoding or decoding is performed by sensor nodes, it is important that they be simple. Fortunately, fountain codes can be encoded using a simple bitwise *xor*, while decoding is performed using a Belief Propagation (BP) decoder.

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 storage nodes finish receiving the source data, they encode

* Corresponding author.
E-mail addresses: louai@kfupm.edu.sa (L. Al-Awami), hossam@cs.queensu.ca (H.S. Hassanein).

http://dx.doi.org/10.1016/j.comnet.2017.02.004
1389-1286/© 2017 Elsevier B.V. All rights reserved.

it and store it locally. The stored data can then be collected from nodes by data collectors in a random fashion. The proposed system enables a distributed implementation of an RSD-based DDSS over wireless network where nodes are resource-constrained and source data is decentralized. The goal of the scheme is to achieve data survivability while attaining simplicity and energy conservation. The proposed scheme is compared to existing schemes using simulation. Existing literature only evaluates the performance of building the data storage, namely dissemination and encoding. A unique aspect to this study is the inclusion of the cost data retrieval as well. The proposed scheme is shown to be superior in dissemination, encoding, storage, and data retrieval costs[1].

Our contributions in this paper include:

- Introducing a decentralized and energy-efficient data dissemination and encoding mechanism for building distributed data storage using LT-codes.
- Developing a random selection policy derived from RSD that can be used in building a distributed storage using local information only.
- Providing comparative study of exemplary schemes form the literature using simulation, and demonstrating how the proposed scheme greatly improves the performance of DDSS compared to other schemes.

The paper is organized as follows. Section 2 lays out some background and related work. The proposed scheme is discussed in Section 3 and evaluation and comparison results are presented in Section 4. Finally, Section 5 concludes the paper. A brief introduction of LT-codes is presented in Appendix A.

2. Background and related work

In this section, we presents some definitions and relevant literature. We discuss concepts related to distributed storage and Fountain codes. We also offer an overview of related work.

2.1. Distributed data storage

A distributed data storage system consists of *native packets, encoded packets, source nodes, storage nodes*, and *data collectors* as depicted in Fig. 1. A source node produces a native packet and disseminates multiple copies to a subset of storage nodes during the dissemination phase. After receiving the native packets from multiple source nodes, the storage nodes encode multiple received packets together to make an encoded packet. During data collection, data collectors contact multiple storage nodes to retrieve encoded packets and decode them to restore the native packets.

When a source node generates a data packet, it can either *store* it locally, *replicate* it over multiple storage nodes, or *encode* it over multiple storage nodes. Storing the data locally is simple, but it does not provide any resilience against data loss. In addition, confining data to a single location reduces its availability, and hinders scalability. Data replication increases data resistance against loss, and can aid in scalability and availability, but it comes at a high cost of storage requirements. Data coding can achieve all goals at a reasonable processing cost when the appropriate codes are used. In fact, it has been shown that using the same level of redundancy, coding can achieve an order of magnitude higher reliability than replication [8]. Unlike encoding-based schemes, replication-based approaches suffer form complicated data gathering protocols. Finally, the cost of retrieving data from a replication-based storage is more than that of a coding-based storage [9].

[1] A shorter version of this paper studying energy efficiency during storage building only was presented previously [7]

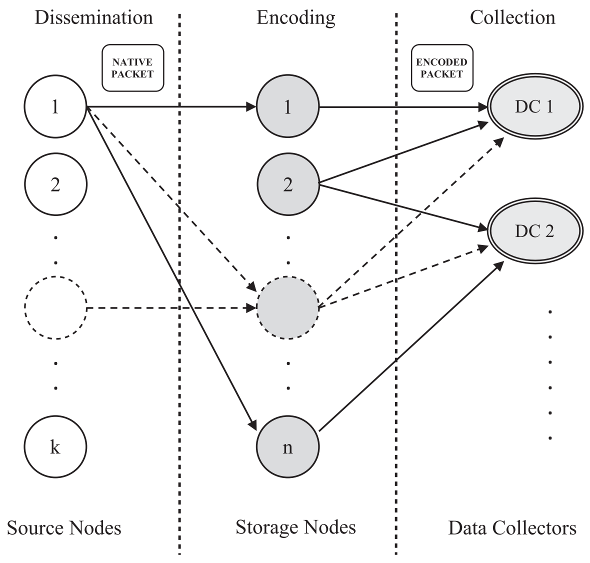

Fig. 1. Distributed data storage model.

2.2. Fountain codes

The concept of fountain codes was first introduced by Byers et al. in 1998 [3]. Later, Luby introduced Luby Transform (LT)-codes [10], the first realization of fountain codes. Other codes have later been introduced including online codes [11] and raptor codes [12]. 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 [13]. Fountain codes enjoy a set of features that make them appealing to our settings. Given a set of k source packets and n encoded packets, the number of packets required for decoding is $(1+\epsilon)k$, where $\epsilon > 0$ is the decoding overhead, using a low complexity 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. We discuss in the next subsection some proposals on how to achieve this. Further discussion on Fountain codes is presented in Appendix A.

2.3. Related work

The topic of implementing distributed data storage has received a lot of attention due to its wide set of potential applications [2,4–6,14–17]. Proposals can be categorized based on the target codes used. The works in [14] and [15] use variants of Random Linear Codes (RLC). While simple to generate, RLC require substantial computational power for both encoding and decoding compared to other siblings in the erasure codes family. This is mainly due to the fact that RLC require a gaussian elimination based decoder which comes with a hefty complexity tag of $\mathcal{O}(n^3)$[18]. To tackle encoding/decoding complexities, other studies have sought aid using fountain codes which are known for their low-complexity encoding and decoding requirements. Based on the fountain codes of choice, the works in that area can be further classified into "LT-codes"-based or "Raptor-codes"-based. The works in [4,5] try to implement an LT-codes over a network of wireless devices in a decentralized manner. In our study, we also use LT-codes to implement a distributed storage system over a set of resource-constrained wireless devices. In contrast, the works in [14] and [17] implement a distributed storage using raptor codes. Since the target of this study is the work in [4] and [5], we explain their work in more details below.

In reference [4], 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 authors assume that each node is aware of its location. They also assume the existence of a Greedy Perimeter Stateless Routing (GPSR) [19] 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 by sending 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 a storage node which can substantially impact the energy requirements due to the two-way communications.

In reference [5], 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 [20], 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 $\pi = \{\pi_1, \pi_2, \ldots\}$, 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). 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.

3. Decentralized robust soliton storage

This section presents the proposed decentralized robust soliton storage (DRSS)

3.1. System model

The scheme assumes a network $N(k, n)$ comprising a set of k *source nodes*, $U = \{u_1, u_2, \ldots, u_k\}$. As with similar distributed storage schemes [5,14,21], we add a set of n redundant *storage nodes* to achieve additional reliability. We denote the set of storage nodes by $V = \{v_1, v_2, \ldots, v_n\}$. 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* [21].

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.

3.2. DRSS operation

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 $Q = \{Q_1, Q_2, \ldots, Q_k\}$, 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 copies of a data packets to be forwarded by each source node. 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 ith entry equals to 1 if the packet from u_i has been used to generate the encoded packet at this node, and 0 otherwise.

We now present how to determine the mapping Q and consequently the selection probability $P(i)$. The case of RSD is discussed next. The case of Ideal Soliton Distribution (ISD) was omitted due to impracticality of use.

3.3. DRSS using Robust Soliton Distribution (RSD)

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,

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

where

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

Now, given an RSD as defined by Eq. A.2, and substituting the values of $\rho(i)$ and $\tau(i)$ then,

$$u(i) = \begin{cases} \frac{1}{Z}\left[\frac{1}{k} + \frac{S}{k}\right], & \text{for } i = 1 \\ \frac{1}{Z}\left[\frac{1}{i(i-1)} + \frac{S}{ik}\right] & \text{for } 1 < i < \frac{k}{S} \\ \frac{1}{Z}\left[\frac{1}{i(i-1)} + \frac{S}{k}\ln\left(\frac{S}{\delta}\right)\right] & \text{for } i = \frac{k}{S} \\ \frac{1}{Z}\left[\frac{1}{i(i-1)}\right] & \text{for } \frac{k}{S} < i \leq n. \end{cases} \tag{3}$$

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

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

Therefor, q_i can be expressed as

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

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

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

where $d(i)$ is the degree of node i. The probabilities of the remaining storage nodes is then determined using the following relation

$$p_i = d(i) \times p_1. \tag{4}$$

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, the RF can be computed as:

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

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, it is 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 the dissemination 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 recovered. 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)
1: $n \leftarrow k \times (s+1)$
2: Generate node degree mapping $Q$ and $d(i)$ for all nodes
3: Build selection probability $P(i)$
4: $m \leftarrow \frac{\sum_{i=1}^{n} d(i)}{k}$
5: for $j = 1 \rightarrow m$ do
6:     Selects target storage node $v_j$ as in $P(i)$
7:     Forward data $x_i$ to $v_j$
8:     $P(i) = \frac{P(v_j)}{\sum_{l} P(l)\ \forall l \neq i}$
9: end for
```

Algorithm 2 DRSS (storage node).

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

4. Performance evaluation

The evaluation of the proposed scheme has been carried out using simulations. We have also implemented the three schemes proposed in references [4] and [5] for comparison. Other similar schemes exist in the literature, but they target other distributions such as raptor codes, e.g. [17,22–24]. We are mainly interested in the collective performance of each scheme in three areas: quality, efficiency, and robustness.

4.1. 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 [25]. There exist other measures of divergence between probability distributions such as KL-divergence [26]. One limitation of KL-Divergence compared to Hellinger distance is that it requires–by definition –all probabilities associated with all 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 $P = \{p_1, p_2, \dots, p_k\}$ and $Q = \{q_1, q_2, \dots, q_k\}$ quantifies the similarity between P and Q. It is defined as

$$H(P, Q) = \frac{1}{\sqrt{2}} \sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2. \quad (5)$$

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.

4.2. 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 we will see shortly.

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 ξ_t, $\xi_r >> \xi_e$. The energy of the data collector is assumed to be infinite.

4.3. 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 operates on a network of $n = (s + 1)k$ nodes, where only k of them are source nodes. To be consistent with other schemes, we have chosen a grid topology. The generation of the location of nodes on the network is chosen uniformly at random.

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), 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 2, 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. For the purpose of our comparison, 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 3 shows the number of operation required by each scheme for different network sizes. The operations of interest that contribute to the energy are the 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

Table 1
A summary of different mathematical symbols used.

Attribute	Meaning
k	Number of source nodes
n	Number of storage nodes
s	Data survivability
$N(k, n)$	Network of k source and n storage nodes
U	Set of source nodes
u_i	Source node i
V	Set of storage nodes
v_i	Storage node i
$d(i)$	Degree of node i
Q_i	Subset of storage nodes with degree i
$P(i)$	Selection probability for nodes with degree i
m	Redundancy factor
q_i	Index of storage nodes
$\rho()$	*pdf* of soliton distribution
S	Spike value for RSD
$u(i)$	RSD pmf
δ	RSD failure probability
p_i	Selection probability for node with degree i
Z	Normalization factor for RSD
$H(P, Q)$	Hellinger distance between P and Q
ζ_s	Energy consumed by sensing
ζ_t	Energy consumed by transmitting
ζ_r	Energy consumed by receiving
ζ_e	Energy consumed by encoding
Ξ	Total energy required
P_s	Probability of successful decoding
β_0	Average number of void packets
β	Decoding overhead
Ω	Divergence-Efficiency-Robustness Indicator
r	Data retrieval ratio

Table 2
Hellinger distance (H) for DRSS, EDFC, ADFC, and DF (95% Std. Err. = ±0.001).

k	n	Hellinger distance			
		DRSS	EDFC	ADFC	DF
50	169	0.114	0.251	0.274	0.190
75	256	0.105	0.234	0.277	0.168
100	344	0.102	0.236	0.330	0.171
125	400	0.096	0.239	0.450	0.173
150	484	0.092	0.230	0.433	0.161

Table 3
Power required to build RSD storage.

k	n	Operation	DRSS	EDFC	ADFC	DF
50	169	COD (ζ_e)	683	576	614	676
		SND (ζ_t)	3967	11,021	9746	11,069
		REC (ζ_r)	3967	11,021	9746	11,069
Total power (Ξ)[*in watts*]			215.65	592.50	524.48	595.57
95% Std. err. of mean			± 4.063	± 15.22	± 7.373	± 36.01
75	256	COD (ζ_e)	1068	897	919	1084
		SND (ζ_t)	7887	18,074	15,222	20,717
		REC (ζ_r)	7887	18,074	15,222	20,717
Total power (Ξ)[*in watts*]			427.29	971.44	818.97	1113.78
95% Std. err. of mean			± 5.163	± 24.05	± 9.594	± 69.71
100	344	COD (ζ_e)	1485	1276	1294	1551
		SND (ζ_t)	11750	24,988	18,819	31,449
		REC (ζ_r)	11750	24,988	18,819	31,449
Total power (Ξ)[*in watts*]			636.05	1343.24	1013.29	1690.27
95% Std. err. of mean			± 7.761	± 27.75	± 14.42	± 96.35
125	400	COD (ζ_e)	2089	1739	1793	2105
		SND (ζ_t)	18,626	33,425	24,879	47,091
		REC (ζ_r)	18,626	33,425	24,879	47,091
Total power (Ξ)[*in watts*]			1006.94	1796.93	1349.00	2529.89
95% Std. err. of mean			± 8.063	± 33.32	± 20.41	± 131.8
150	484	COD (ζ_e)	2492	2093	2128	2592
		SND (ζ_t)	24,642	40,320	29,881	62,635
		REC (ζ_r)	24,642	40,320	29,881	62,635
Total power (Ξ)[*in watts*]			1330.81	2167.59	1609.27	3363.93
95% Std. err. of mean			± 10.15	± 39.22	± 22.56	± 173.5

Table 4
Void packets ratio (β_0) for DRSS, EDFC, ADFC, and DF.

k	n	No. void packets (β_0)			
		DRSS	EDFC	ADFC	DF
50	169	0.5(0)	2.06(0.041)	4.46(0.089)	0(0)
75	256	0.04(0)	2.44(0.032)	6.24(0.083)	0(0)
100	344	0.09(0)	2.24(0.022)	12.61(0.126)	0(0)
125	400	0.01(0)	2.24(0.018)	18.07(0.145)	0(0)
150	484	0.03(0)	3.06(0.020)	22.46 (0.149)	0(0)
95% Std. Err		0	± 0.18	± 0.15	0

total energy required for code construction including data dissemination and encoding. Ξ will be used later to compare the schemes under consideration.

The third metric that must be considered is the ratio of encoded packets with degree zero. 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 helps quantify the fraction of storage nodes that were not reached by the dissemination mechanism. Table 4 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.

To see the impact of void packets on the robustness of the code, let's consider the probability of successful decoding (P_s). Fig. 2 shows that as the number nodes failing increases, DRSS maintains a medium P_s indicating that code survivability is between that of other schemes. 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 is. This does not necessarily translate to higher survivability. Moreover, EDFC's random walk mechanism results in more localized dissemination of data, i.e. random walk does not distributed the data over the network uniformly. Hence, when failure

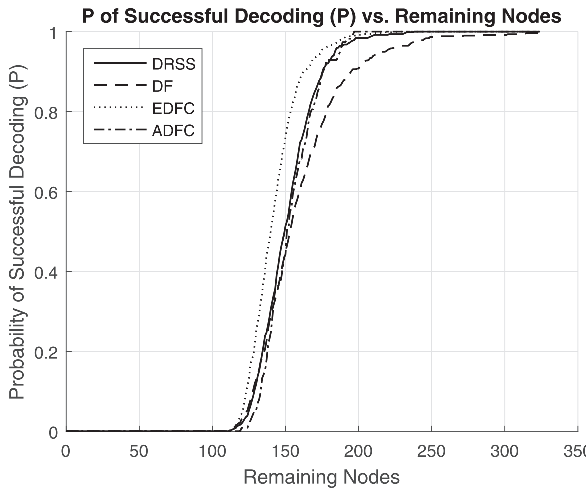

Fig. 2. Probability of successful decoding (P_s): $k = 100$, $s = 3$, $\mathbb{F}_2$.

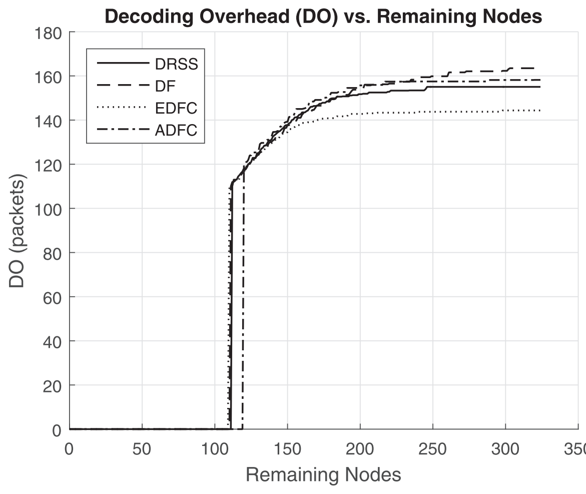

Fig. 3. Decoding overhead (β): $k = 100$, $s = 3$, $\mathbb{F}_2$.

Table 5
DER (Ω) for DRSS, EDFC, ADFC, and DF.

k	n	DER			
		DRSS	EDFC	ADFC	DF
50	169	348	765	715	708
75	256	490	1230	1114	1301
100	344	758	1690	1475	1979
125	400	1113	2259	2152	2968
150	484	1492	2709	2546	3906

occurs in certain region of the network, the code may not be decodable. Fig. 3 shows the decoding overhead for the four schemes.

We finally, present the Divergence Efficiency Robustness Indicator (DER) (Ω) to be a cost function to compare the three schemes by capturing all the three aspects at once. DER is defined as

$$\Omega = \Xi(1 + \beta_0 + H). \tag{6}$$

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 5 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

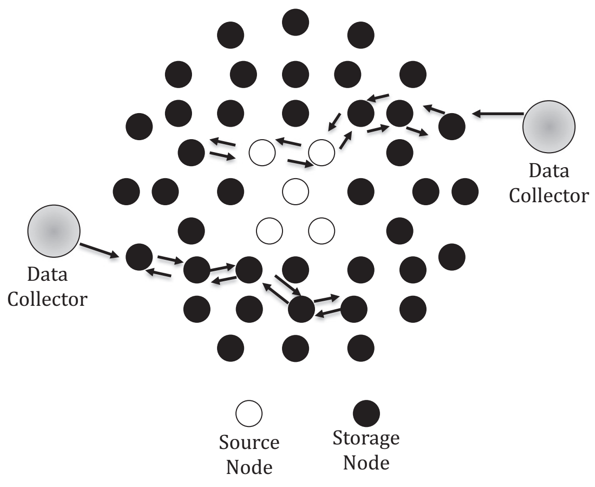

Fig. 4. Data collection network model.

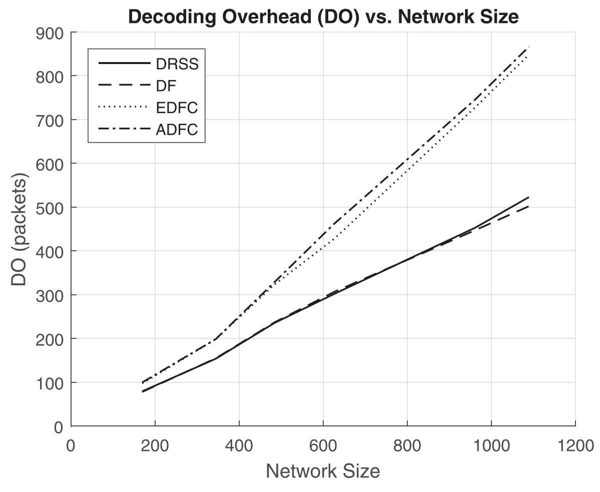

Fig. 5. Data packets required vs. network size.

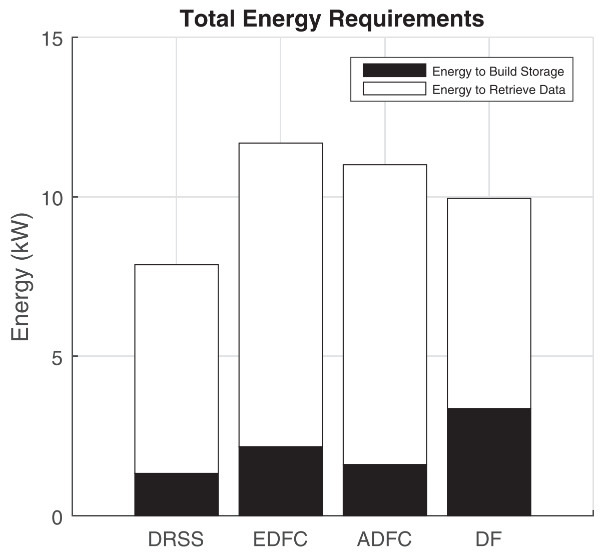

Fig. 6. Total power required for building data storage and retrieval (k=150, r=100).

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 almost twice the energy than DRSS, EDFC and ADFC still require substantially more, due to the convergence characteristics of the random walk.

4.4. Energy requirements of data retrieval

Decentralized storage schemes strive to guarantee data survivability. However, data is usually not meant to stay in the network forever and would eventually be collected for processing. Data collection involves contacting storage nodes to retrieve encoded packets. This obviously requires energy and must be considered when comparing different storage schemes.

In order to evaluate the energy cost of data collection we use the network model depicted in Fig. 4. We assume data collectors to show up at a random locations and contact their nearest storage node. When a storage node receives a data collector request, it replies by sending back its encoded packet. If the data collector has enough data to decode the original data packets, it signals decoding completion. Otherwise, the storage node contacts a random neighbor which repeats the same operation. Note that the data collection protocol is not meant to be optimal nor is it complete. The sole purpose of the protocol is to compare the number of packets required to retrieve the full set of data packets by the data collector, from storage built using the different four schemes.

There are two factors that affect the performance of each scheme during data collection. The first is the degree distribution exhibited by the encoded data. The closer the degree distribution of the data storage to RSD, the less the number of packets required for decoding. The second factor is the geographical distribution of data packets over the storage network. It is favorable to have data packets uniformly distributed over the network. This results in an equal likelihood to retrieve each packet from anywhere in the network.

We study two metrics to compare the four schemes. The first metric is the average number of packets required to complete the decoding. The second is the total energy spent on data collection.

Fig. 5 shows the average number of packets required for decoding for each scheme. It is apparent that the schemes DRSS and DF require less packets to be collected. than the other two random walk-based schemes, namely EDFC and ADFC. This can be explained by two facts. First, both DRSS and DF achieve better approximations to RSD. Consequently, they perform better when decoded using BP decoder. The second factors relates to the geographical distribution of the data. Both EDFC and ADFC are expected to have a more localized geographical dissemination due to the nature of random walks. On the other hand, both DRSS and DF do not consider storage node location during data dissemination and thus result in a more uniform geographical distribution.

Proportional to the number of data packets is the energy needed for data collection. We define the retrieval ration (r) to be the number times each source packet is retrieved. The idea behind the retrieval ratio is to store-once and use-many-times, before the data is replaced with new one. Fig. 6 compares the total energy needed by each scheme for building the storage and retrieving the data from the storage. We assume $r = 100$. It is important to note that while DF achieves the best results for data collection, it requires the most energy during data dissemination. In addition, the difference in the required energy for data collection between DF and DRSS is extremely small, especially for networks of less than 1000 nodes. When combined with the observation that DRSS data dissemination requirements are the least, it clear that DRSS outperforms all other schemes.

5. 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 compared the new scheme to three similar schemes in the literature; DF, EDFC, and ADFC; and show that DRSS achieves the required degree distribution with a fraction of the energy required by the other schemes even in small settings. In addition, retrieval of data from a storage built with DRSS costs significantly less. We strongly believe that this can prove the potential and the suitability of LT-codes to provide data survivability in future applications of large-scale resource-constrained wireless networks and enable more reliable operations.

Acknowledgements

The authors would like to thanks King Fahd University for Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, for the generous support. This research is also supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number: CRDPJ445517. We extend our thanks to Dr. Shahram Yousefi for the fruitful comments and discussions.

Appendix A. LT codes and Robust Soliton Distribution (RSD)

To provide the reader with a better understanding of our work, we present a brief description of the operation of LT-codes and it's degree distribution, i.e. RSD. Then, we describe an analysis of the performance of the RSD. This background is essential to the discussion on the development of the proposed scheme.

A1. LT codes

LT-codes were one of the first realization of rateless codes. The encoder generates packet degrees according to the RSD while the decoder uses the BP algorithm. In this section we discuss both the encoder and decoder of LT-codes.

A1.1. Encoder

Given a file to transmit, the LT encoder divides the file into k equal parts, referred to as *source packets*. The decoder of an LT-codes then 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*. Let $X_i = \{x_1, x_2, \ldots, x_d\}$ be the set of source packets chosen by the encoder at step i, then an encoded packet e_i can be generated as

$$e_i = x_{i_1} \oplus x_{i_2} \oplus \ldots x_{i_d} \forall x_{i_j} \in X_i.$$

d is referred to as the *packet degree*. The sets of source and storage packets can be represented as a bipartite graph, where each encoded packet is connected to the source packets used to generate it. In such a representation, the set of source packets that are connected to the same encoded packet are called *neighbours*.

A1.2. Decoder

The design of LT-codes is geared towards the Belief Propagation (BP) decoder. The BP decoder works iteratively. At each step, the decoder searches for encoded packets with degree one. The set of all degree one packets at the current step is 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 causing the next iteration to start. The process of generating new degree one packets is referred to as *release* of packets. The decoder repeats this process until all encoded packets are decoded.

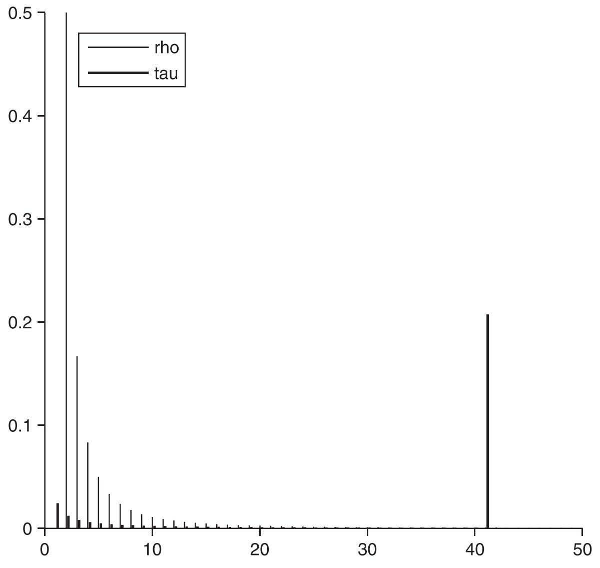

Fig. A.7. $\rho(i)$ and $\tau(i)$ for $k = 10,000$, $c = 0.2$, and $\delta = 0.05$ [27].

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 it is 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 $n\log(n)$.

A2. Robust Soliton Distribution (RSD)

As discussed earlier, a key aspect of the design of fountain codes is the degree distribution $\rho(i)$. In the case of LT-codes, RSD is used. The distribution can be described by the first introducing the probability mass function (pmf) of the ISD:

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

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 exists.

Therefore, Luby in reference [10] proposed the RSD that is used as the degree probability distribution. RSD is defined by the following pmf

$$u(i) = \frac{\rho(i) + \tau(i)}{Z}, \tag{A.2}$$

where

$$Z = \sum_i \rho(i) + \tau(i) \tag{A.3}$$

and

$$\tau(i) = \begin{cases} \dfrac{S}{i \times k}, & \text{for } i = 1, 2, \ldots, \dfrac{k}{S} - 1 \\ \dfrac{S}{k} \ln\left(\dfrac{S}{\delta}\right) & \text{for } i = \dfrac{k}{S} \\ 0 & \text{for } i > \dfrac{k}{S}, \end{cases} \tag{A.4}$$

where

$$S = c \ln\left(\frac{k}{\delta}\right)\sqrt{k}. \tag{A.5}$$

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

Fig. A.7 shows an example of RSD for $k = 10,000$. The intuition behind RSD is to modify the ISD to also include enough packets of higher degrees to guarantee covering all the source packets. This is shown in Fig. A.7 by the spike at $\frac{k}{S} = 41$.

References

[1] K. Xu, H. Hassanein, G. Takahara, Q. Wang, Relay node deployment strategies in heterogeneous wireless sensor networks, Mob. Comput. IEEE Trans. 9 (2) (2010) 145–159, doi:10.1109/TMC.2009.105.

[2] M. Albano, S. Chessa, Replication vs erasure coding in data centric storage for wireless sensor networks, Comput. Netw. 77 (C) (2015) 42–55, doi:10.1016/j.comnet.2014.11.018.

[3] J.W. Byers, M. Luby, M. Mitzenmacher, 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, ACM, New York, NY, USA, 1998, pp. 56–67, doi:10.1145/285237.285258.

[4] A.G. Dimakis, V. Prabhakaran, K. Ramchandran, Distributed fountain codes for networked storage, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006), 2006, pp. 1149–1152, doi:10.1109/ICASSP.2006.1661484. Toulouse, France.

[5] Y. Lin, B. Liang, 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, doi:10.1109/INFCOM.2007.194. Anchorage, AK, USA.

[6] S.A. Aly, Z. Kong, E. Soljanin, Fountain codes based distributed storage algorithms for large-scale wireless sensor networks, in: 7th International Conference on Information Processing in Sensor Networks (IPSN '08), IEEE Computer Society, Washington, DC, USA, 2008, pp. 171–182. http://dx.doi.org/10.1109/IPSN.2008.64.

[7] L. Al-Awami, H.H. Hassanein, Energy efficient distributed storage systems with LT-codes in resource-limited wireless systems, in: 2015 IEEE Global Communications Conference (GLOBECOM), 2015, pp. 1–6, doi:10.1109/GLOCOM.2015.7417731. San Diego, CA, USA.

[8] H. Weatherspoon, J. Kubiatowicz, Erasure coding vs. replication: a quantitative comparison, in: First International Workshop on Peer-to-Peer Systems (IPTPS '01), Springer-Verlag, London, UK, 2002, pp. 328–338.

[9] S. Acedan'ski, S. Deb, M. MĘdard, R. Koetter, How good is random linear coding based distributed networked storage, in: First Workshop on Network Coding, Theory, and Applications (NetCod 2005), 2005, pp. 1–6. Riva del Garda, Italy.

[10] M. Luby, LT Codes, in: 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 271–280, doi:10.1109/SFCS.2002.1181950.

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

[12] A. Shokrollahi, Raptor codes, IEEE Trans. Inf. Theory 52 (6) (2006) 2551–2567, doi:10.1109/TIT.2006.874390.

[13] D. Project, DVB-IPTV 1.4: Transport of MPEG 2 TS Based DVB Services over IP Based Networks (and associated XML) (dts 102 034 v1.4.1), 2009, (http://dvb.org/technology/standards/index.xml).

[14] A.G. Dimakis, V. Prabhakaran, K. Ramchandran, Decentralized erasure codes for distributed networked storage, IEEE/ACM Trans. Netw. 14 (SI) (2006) 2809–2816. http://dx.doi.org/10.1109/TIT.2006.874535.

[15] L. Al-Awami, H.S. Hassanein, Distributed data storage systems for data survivability in wireless sensor networks using decentralized erasure codes, Comput. Netw. 97 (2016) 113–127. http://dx.doi.org/10.1016/j.comnet.2016.01.008.

[16] A.G. Dimakis, K. Ramchandran, Network coding for distributed storage in wireless networks, in: Networked Sensing Information and Control, 2, Springer US, 2008, pp. 115–134.

[17] S. Aly, K. Zhenning, E. Soljanin, Raptor codes based distributed storage algorithms for wireless sensor networks, in: IEEE International Symposium on Information Theory (ISIT 2008), 2008, pp. 2051–2055, doi:10.1109/ISIT.2008.4595350. Toronto, ON, Canada

[18] G. Garrammone, On decoding complexity of reed-solomon codes on the packet erasure channel, IEEE Commun. Lett. 17 (4) (2013) 773–776, doi:10.1109/LCOMM.2013.021913.122427.

[19] B. Karp, H.T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: 6th Annual International Conference on Mobile Computing and Networking (MobiCom '00), 2000, pp. 243–254, doi:10.1145/345910.345953. New York, NY, USA.

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

[21] L. Al-Awami, H. Hassanein, Data survivability for WSNs via decentralized erasure codes, in: Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International, 2012, pp. 94–99. Limassol, Cyprus.

[22] Z. Kong, S.A. Aly, E. Soljanin, Decentralized coding algorithms for distributed storage in wireless sensor networks, IEEE J. Sel. Areas Commun. 28 (2) (2010) 261–267. http://dx.doi.org/10.1109/JSAC.2010.100215.

[23] C. Stefanovic, V. Stankovic, M. Stojakovic, D. Vukobratovic, Raptor packets: a packet-centric approach to distributed raptor code design, in: IEEE International Symposium on Information Theory (ISIT 2009), 2009, pp. 2336–2340, doi:10.1109/ISIT.2009.5205950.

[24] D. Vukobratovic, C. Stefanovic, V. Crnojevic, F. Chiti, R. Fantacci, A packet-centric approach to distributed rateless coding in wireless sensor networks, in: 6th IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '09), 2009, pp. 1–8. Roma, Italy.

[25] E. Hellinger, Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen., Fõr Die Reine Und Angewandte Mathematik (1909).

[26] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Stat. 22 (1951) 79–86.

[27] D.J.C. Mackay, Fountain codes, IEE Commun. 152 (2005) 1062–1068.

Louai Al-Awami is currently an Assistant Professor at the Computer Engineering Department, at King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He earned his BSc and MSc in Computer Engineering from the Computer Engineering, at KFUPM, in 2002 and 2006, and his Ph.D. from the Electrical and Computer Engineering Department at Queens University, Canada, respectively. Louai has also taught many courses and laboratories while working as a lecturer at KFUPM. Louais research interest include wireless sensor networks data reliability; distributed storage systems, network coding and data dissemination, and information centric networking. Louai is actively engaged in the IEEE member since 2002.

Hossam S. Hassanein is a leading authority in the areas of broadband, wireless and mobile networks architecture, protocols, control, and performance evaluation. His record spans more than 500 publications in journals, conferences and book chapters, in addition to numerous keynotes and plenary talks in flagship venues. Dr. Hassanein has received several recognition and best paper awards at top international conferences. He is also the founder and director of the Telecommunications Research Lab (TRL) at Queen's University School of Computing, with extensive international academic and industrial collaborations. He is a fellow of the IEEE, and is a former chair of the IEEE Communication Society Technical Committee on Ad hoc and Sensor Networks (TC AHSN). Dr. Hassanein is an IEEE Communications Society Distinguished Speaker (Distinguished Lecturer 2008–2010).