Computer Networks 97 (2016) 113–127

Contents lists available at ScienceDirect

Computer Networks

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

Distributed Data Storage Systems for Data Survivability in Wireless Sensor Networks using Decentralized Erasure Codes

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

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

ARTICLE INFO

Article history:
Received 10 February 2015
Revised 14 January 2016
Accepted 19 January 2016
Available online 26 January 2016

Keywords:
Wireless Sensor Network
Data Survivability
Decentralized Erasure Codes
Network Coding
Distributed Data Storage

ABSTRACT

Achieving reliability in Wireless Sensor Networks (WSNs) is challenging due to the limited resources available. In this study, we investigate the design of data survivability schemes using decentralized storage systems in WSNs. We propose a data storage system design based on Decentralized Erasure Codes (DEC) that features a simple and decentralized construction of the target code. The proposed framework allows sensor nodes to cooperate to build an erasure code-based storage that can tolerate a given failure/erasure rate. Code construction and decoding can both be performed randomly allowing for a distributed operation with no prior setup or coordination between source nodes. Further, we present two approaches that utilize Random Linear Network Coding (RLNC) to enhance the proposed scheme in order to achieve energy efficiency. We present the theoretical basis of the schemes then validate and evaluate their performance through simulations.

1. Introduction

Wireless Sensor Network (WSN) technology is being increasingly deployed in a diverse range of applications. Intelligent Transportation Systems (ITSs) [1], Smart Grids [2], and the Internet of Things (IoT) [3] are just a few examples of technologies where WSNs are used. Generally, WSNs are comprised of **sensor nodes** that are equipped with one or multiple sensors, a processing unit, and a wireless communication module. Sensor nodes cooperate in monitoring a phenomenon of interest and in relaying the sensed data to a **sink node** for processing. When produced in large numbers, sensor nodes can be extremely inexpensive, and hence they can be deployed in greater numbers to build large scale networks. WSNs have stringent constraints, especially regarding power consumption and scalability. Furthermore, reliability becomes a key requirement for WSNs when deployed in unattended applications or under harsh working conditions.

To preserve the sensed data captured by sensor nodes, WSNs nodes can benefit from using Distributed Data Storage Systems (DDSSs) technology. Data storage systems represent an essential component of today's networks and they have been researched for a long time. Lately, data storage technology is being revisited especially in the contexts of Content Centric Networking (CCN) [4] and cloud computing [2]. DDSSs utilize *hardware redundancy* and *data replication* to protect data in case of possible failures. More specifically, given a data packet, a DDSS replicates the packet over multiple physical storage devices, such that when a subset of these devices fails, the data packet can be retrieved from the surviving ones.

In this study, our goal is to design a DDSS that is tailored for WSNs data reliability applications. For that, we first introduce the notion of data survivability as a quantitative parameter that links the amount of redundancy required to the maximum failure that can be tolerated. We

* Corresponding author at: School of Computing, Queen's University, Kingston, ON, Canada. Tel.: +1 6135336336.
E-mail addresses: louai@kfupm.edu.sa, louai@cs.queensu.ca (L. Al-Awami), hossam@cs.queensu.ca (H.S. Hassanein).

http://dx.doi.org/10.1016/j.comnet.2016.01.008
1389-1286/

then show how data survivability can be useful by implementing a data survivability scheme, called Decentralized 30 Erasure Codes for Data Survivability (DEC-DS). DEC-DS is based on Decentralized Erasure Codes (DEC) [5–7]. Besides being decentralized, DEC has a predictable algebraic structure allowing for quantifiable performance. After that, we present two methods to enhance the energy efficiency of DEC-DS by exploiting Network Coding (NC). The two schemes are referred to as DEC Encode-and-Forward (DEC-EaF) and DEC Encode-and-Disseminate (DEC-EaD). NC [8] has emerged as an information-theoretic tool and has been shown to decrease energy consumption and complexity while increasing throughput and reliability [9]. Random Linear Network Coding (RLNC) [10] has been later proposed as a practical implementation of Network Coding. In this study, we utilize RLNC to increase the efficiency of the proposed storage system by reducing communication overhead and consequently energy requirements. The main contributions of this paper are introducing the notion of data survivability and presenting the three data storage schemes, DEC-DS, DEC-EaF, and DEC-EaD.

The remainder of the paper is organized as follows. In Section 2, we present some background material and review related work. The proposed data survivability framework is discussed in Section 3. Section 4 shows two schemes using RLNC to improve the efficiency of the proposed data survivability application. Experiments and results are discussed in Section 5. Finally, Section 6 concludes the paper. Some important results from the theory of random matrices over finite fields, which will be used in designing the codes, are presented in Appendix A.

2. Background and related work

Before we discuss the proposed schemes, we present the advantages and disadvantages of replication and encoding-based storage. We then present the concept of data survivability and how it differs from network survivability. We also present an overview of Fountain Codes and DEC; and survey related literature on DDSSs in WSNs.

2.1. Replication Vs. encoding

Replicated data can be stored either as is (*replication-based storage*) or encoded using erasure codes (*coding-based storage*). Coding-based solutions can achieve many advantages over replication-based solutions at a slight increase in processing cost. Unlike coding, replication often requires more storage space on every storage node. In other words, to attain the same level of reliability, replication-based schemes require more redundancy than coding-based schemes. In fact, for the same level of redundancy, coding can achieve an order of magnitude higher reliability than replication [11]. In addition, replication-based approaches also need to keep track of where each data exist, resulting in complicated data gathering protocols. Moreover, it has been shown analytically that on average the number of data blocks needed to reconstruct a complete data set from a replication-based distributed storage is more than what is needed when using coding-based distributed storage [12].

2.2. Data Survivability vs. Network Survivability

As aforementioned, WSNs combine a set of unique requirements such as limited energy, dense deployment, and harsh working conditions. Consequently, developing a DDSS for WSNs needs to tackle such requirements. To address data reliability, sensor data in WSNs need to be maintained using a reliability mechanism. This is especially important when a sink node is not available, such as in the case of Delay Tolerant Networks (DTNs). In this regard, we present data survivability as a design parameter that describes the required data resilience against failures. We make a distinction between data and network survivability. *Network survivability* [13] focuses on using redundancy as a means to guarantee network continuity in case of nodes failure. *Data survivability* provides a means to prevent loss of data in the network in case of failure through the use of redundancy. Also, while network survivability requires redundancy in hardware and software, data survivability utilizes redundancy in storage and data. Other similar concepts exist in the literature such as "*service survivability*" which focuses on continuity of the service even when the physical system fails, through using backup servers [14].

2.3. Fountain Codes

There exists some resemblance between Decentralized Erasure Codes (DEC) and Fountain Codes. Therefore, we provide a brief description of Fountain Codes to lay the ground for the discussion on DEC. The literature on DDSSs contains some overlap between the two codes. We believe it is useful to discuss the two families and show why DEC is better suited for data survivability.

Since their introduction in late 1990's, Fountain codes [15] have attracted an increasing interest in the research community. The main attracting attribute of this family of codes is that they are *rateless*, meaning they do not have a fixed rate associated with them a priori. Hence, compared to ordinary erasure codes such as Reed–Solomon Codes [16], rateless codes can adapt to any given erasure channel with an associated erasure probability p_e on-the-fly.

Given a set of k *native* data blocks of equal length $B = \{b_1, b_2, \ldots, b_k\}$ and a probability distribution $\rho(k)$, the encoder of a Fountain code generates n encoded packets as follows. To generate the ith encoded packet, the encoder samples $\rho(k)$ for a value $1 \le d_i \le k$. Then, it uniformly selects d_i random data blocks from B and *xor's* the blocks linearly together under the mathematics of $\mathbb{F}_2$ generating an *encoded* block e_i. d_i is referred to as the *degree* of the encoded block e_i. Similarly, $\rho(k)$ is called the code *degree distribution*. In addition to the encoded block, a k-dimensional binary *encoding* vector $G_i = \{g_{i1}, g_{i2}, \ldots, g_{ik}\}$ is appended to e_i; where every entry g_{ij} is set to 1 if b_j was used to construct e_i and 0 otherwise. g_{ij} is referred to as an *encoding coefficient*. Let $E = \{e_1, e_2, \ldots, e_n\}$ and $G = \{G_1, G_2, \ldots, G_n\}$ be the set of encoded blocks and encoding vectors, respectively. In general, $k < n$. The decoder on the receiving side, keeps receiving encoded blocks until solving the system of linear equations $E_{1\times n} = B_{1\times k}G_{k\times n}$, for B. The number of packets required for decoding beyond k is referred to as *code overhead*. Generally, the decoder requires $n = (1+\epsilon)k$

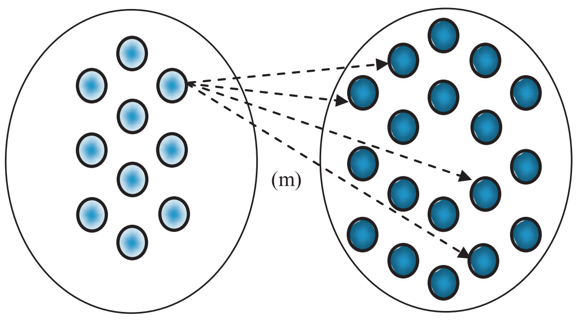


Fig. 1. Source and storage networks ($k = 10$, $n = 18$, and $m = 4$).

encoded blocks to recover all native data blocks, where $\epsilon > 0$.

The key design aspect of rateless codes is the degree distribution $\rho(k)$. In [17], Luby proposed LT codes using the *Robust Soliton Distribution*. For a probability of successful decoding $1 - \delta$, LT codes require an overhead of only $\mathbf{O}(\sqrt{k}\log_2(k/\delta))$ with decoding complexity of $\mathbf{O}(k\log_2(k/\delta))$. Raptor codes [18], on the other hand, achieve a linear encoding and decoding at the expensive of extra overhead ($\mathbf{O}(k)$) using the idea of pre-coding. Note that classic erasure codes such as Reed–Solomon Codes have an encoding and decoding complexities of $k(n-k)\log(n)$.

Despite their advantages, Fountain codes have been designed assuming all source data exist in one location, and hence they are not straightforward to implement when the source data are decentralized.

2.4. Decentralized Erasure Codes (DEC)

DEC was introduced by Dimakis et al. [6]. The basic operation of DEC is illustrated in Fig. 1. Given a network of k source nodes and n storage nodes, where $k < n$, each source node j generates a single data block b_j and forwards it to m storage nodes that are selected uniformly at random. Upon receiving the data blocks, each storage node i generates a random coefficient g_{ij} for every block b_j received, which is drawn from a finite field $\mathbb{F}_q$ for every data block received and combines the received blocks as follows:

$$e_i = (g_{i1}b_1) \oplus (g_{i2}b_2) \oplus \cdots \oplus (g_{ik}b_k).$$

Note that every storage node will receive a different set of data packets, and also that $g_{ij} \neq 0$ if b_j was included in the encoding at the current storage node and 0 otherwise. The storage node then stores the *encoded block* e_i in addition to the k-dimensional vector of random coefficients $G_i = \{g_{i1}, g_{i2}, \ldots, g_{ik}\}$ used for encoding. To retrieve the original blocks, the decoder needs to collect $(1+\epsilon)k$ encoded blocks to solve for B in the system of linear equations $E_{1\times n} = B_{1\times k}G_{k\times n}$, where E represents the vector of encoded data, G represents the matrix of encoding coefficients, and B is the vector of native data.

Despite its similarity to Fountain codes [15,19], DEC is quite different. Whereas in Fountain codes d_i, the degree of each encoded block (the number of packets used to generate a block) can be generated exactly to match $\rho(k)$, DEC has no means of controlling the distribution of the degree of the encoded blocks since the generation of encoded blocks is distributed. We argue that Fountain codes are not suitable for situations such as in WSNs where source data are decentralized, which makes achieving the required degree distribution unpractical. Besides, if decoding is performed off-line such as in the case of DTN, low complexity decoding can be exploited for the sake of longer network life.

In a distributed storage setup, DEC generates a different overhead on the encoder side compared to the overhead seen by the decoder, where overhead is defined as the number of blocks beyond k needed for decoding. Therefore, we define α and β to be the Encoding Overhead (EO) and Decoding Overhead (DO) of DEC, respectively. Generally, $\alpha > \beta$. Let m_i be the number of redundant data packets disseminated by the ith source node. When $m_i = m$ is equal for all source nodes, $\alpha = k(m-1)$. Likewise, we can express $\beta = \epsilon k$.

In [5–7], Dimakis et al. have introduced the idea of DEC and shown that $m \geq 5\frac{n}{k}\log(k)$ is sufficient to guarantee that collecting *any* $(1+\epsilon)k$ encoded blocks is enough to recover the native k blocks with high probability for some $\epsilon > 0$. The decoding is assumed to be using Gaussian Elimination which requires $\mathbf{O}(k^3)$ arithmetic operations or $\mathbf{O}(k^2\log(k))$ when exploiting sparsity of the coefficient matrix. Compared to the original DEC presented in [6], we argue that achieving survivability requires lower *EO* than that required to achieve low delay decoding which comes at the expense of higher *DO*. Fortunately, motivated by the results in Appendix A, we know that the *DO* is upper bounded by $\beta = k + c$ where $c \simeq 8$ for $\mathbb{F}_2$.

The construction of “Robust Soliton Distribution”-like decentralized codes has been investigated in [20–28]. In the *node-centric* approaches [20–24], the source data perform a *random walk* over a set of storage nodes, where at each step, the source data are *xor*-ed with the local data at the current storage node. On the other hand, *packet-centric* approaches [25–28], allow source packets to perform a random walk while encoding from the data on each newly visited storage node, until eventually stopping at the walk-terminating node. The work in [29] studies the suitability of different erasure codes-based in-network storage to different types of networks. There also exists a body of work where distributed encoded storage is used to tackle security and data integrity [30,31]. For a more thorough survey on the topic, see [32].

In this paper, we present a decentralized data survivability scheme (DEC-DS) for WSNs based on DEC. Unlike the original DEC, our objective is to increase the immunity of data to failure rather than reducing the number of packets required for decoding (β) as in [5–7]. Energy in WSNs is crucial and using pure random walk can consume excessive energy to implement. Also, random walk protocols are asymptotic in nature, requiring a large number of nodes to converge to the required distribution. Our approach has been shown to be applicable to networks with as few as ten nodes. In addition, all previous studies assume n, the number of redundant nodes, to be given. We provide a

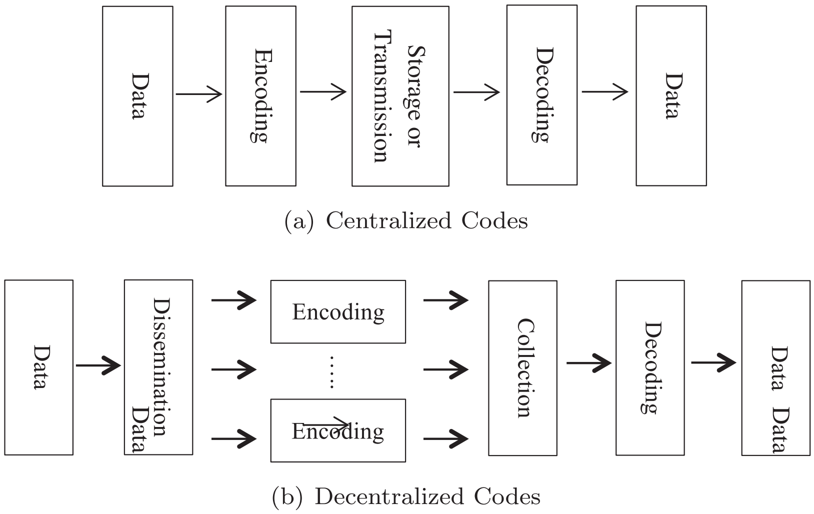


Fig. 2. Centralized vs. Decentralized Erasure Codes.

way to calculate the number of storage nodes needed to achieve the required survivability. We also assume all encoding is performed over $\mathbb{F}_2$. The choice of field has a tremendous impact on encoding/decoding performance in practice [33], and binary encoding can provide simplicity when compared to higher fields. To our knowledge, such survivability scheme does not exist in the literature.

3. Decentralized Erasure Codes for Data Survivability (DEC-DS)

The design of an erasure code in a centralized setup is quite different than that in a decentralized one. The difference is illustrated in Fig. 2. In a centralized code (Fig. 2(a)), all k native data blocks are available at a single encoder. Hence, when sampling a degree d from the degree distribution $\rho(k)$, d can be exactly matched since all k native packets are available to the encoder. In other words, if the degree sampled from the distribution is d, the encoder has all the k data packets where $k \geq d$ packets to generate a packet with the same degree d. However, due to the dissemination phase in Fig. 2(b), the encoder at each node may have only a subset of the k native blocks which could be less than d. Therefore, the required degree d may not be exactly matched. Mathematically speaking, in the centralized case (Fig. 3(a)) the degree of each **row** (the number of non-zero entries) in the matrix (G) can be produced exactly to match the random value d generated by the degree distribution $\rho(k)$. On the other hand, in the case of decentralized codes (Fig. 3(b)), each source node chooses which storage nodes receive its data by disseminating m duplicate copies to m distinct storage nodes, resulting in setting exactly m entries per **column**.

Due to the restriction imposed by the observation made above, we need to resort to a different approach than that used in centralized codes to control the distribution (density) of the matrix G and consequently its properties. For this reason, we manipulate the overall distribution of the matrix instead of manipulating the degree of each row.

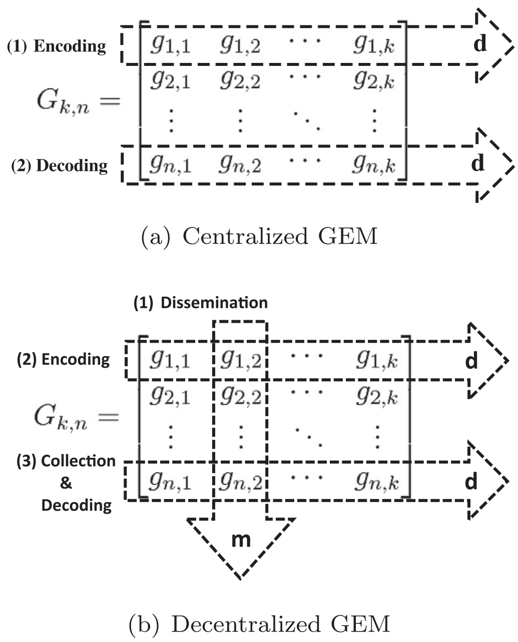


Fig. 3. Encoding Coefficients Matrix for Centralized vs. Decentralized Erasure Codes.

One way to generate the exact distribution of a centralized erasure code is to send *all* source data to *all* storage nodes ($m = n$) and let each storage node choose a subset of the source data according to $\rho(k)$. Clearly, the communication requirements of such a solution are $\mathbf{O}(nk)$. The good news is that we can do better since we have control over the distribution of G. In what follows, we show how we can use the results from the theory of random matrices over finite fields, to implement decentralized codes for data survivability applications. Those results are discussed in Appendix A.

Table 1
Mathematical symbols used in the proposed schemes.

Attribute	Meaning
k	Number of source nodes
n	Number of storage nodes
N	Network size
s	Data survivability
$N(k, s)$	Network of k source and survivability s
X	Set of source nodes
x_i	Source node i
Y	Set of storage nodes
y_i	Storage node i
$C(k, s)$	A code with n storage nodes and survivability s
m	Redundancy Factor (RF)
B_R^j	Set of data packets received by node y_j
e_j	Encoded packet at node y_j
B_s	Set of native data packets
b_i	Data packet from source node x_i
α	Encoding Overhead (EO)
Gj	Encoding vector at node y_j
g_{ji}	ith Encoding coefficient at node y_j
G	Global generating matrix
$\overline{G}$	Local generating matrix
E	Encoded data matrix
P	Invertibility probability

To help the reader follow the description, we have summarized the different notations used in Table 1. Consider a network $N(k, s)$, similar to the one in Fig. 1, with a set X of k **source nodes**, $X = \{x_1, x_2, \ldots, x_k\}$ and a required survivability s. *Survivability* is defined as the maximum fraction of sensor nodes that can fail without compromising the recoverability of the native data. For example, $s = 0.8$ corresponds to a code that can tolerate $s \times 100 = 80\%$ failure ($0.8 \times N$ nodes), where N is the network size. We are interested in designing a storage network with a code $C(k, s)$ and survivability s.

Let n be the number of **storage nodes**. Therefore, $N = k + n$. n can be calculated as

$$n = k(s + 1). \tag{1}$$

Let $Y = \{y_1, y_2, \ldots, y_n\}$ be the set of storage nodes. In addition to storage, nodes in Y are also assumed to serve as relays. We assume that a multi-hop routing mechanism is in place. Each source node x_i generates a data block b_i, then selects a set of m storage nodes $Z = \{z_1, z_2, \ldots, z_m\}$ uniformly and randomly where $Z \subseteq Y$, and sends b_i to the set of selected nodes. We refer to m as the *Redundancy Factor* (RF). Let $B_S = \{b_1, b_2, \ldots, b_k\}$ represent the set of all native packets generated by all the k source nodes. We define the EO as

$$\alpha = (k - 1)m. \tag{2}$$

Upon receiving a set of data blocks B_R^j, each storage node y_j combines the received blocks linearly to generate an encoded block e_j as

$$e_j = b_1 \oplus b_2 \oplus \ldots \;\; \forall\, b_i \in B_R^j,$$

where e_j represents a linear combination of a sum of a random subset of B_S. The number of packets $d_j = |B_R^j|$ used to construct an encoded packet e_j is called the *packet degree* of e_j . Along with e_j, a k-dimensional binary vector $G_j = \{g_{j1}, g_{j2}, \ldots, g_{jk}\}$ is generated with entries as

$$g_{ji} = \begin{cases} 0, & \text{if } b_i \notin B_R^j \\ 1, & \text{otherwise}. \end{cases} \tag{3}$$

G_j is referred to as the *encoding vector*. Each node is assumed to have a storage space for only one encoded packet and its corresponding encoding vector. Let G be the *global generating matrix* as seen in Fig. 3(b). When data are being collected from the network, a subset of storage nodes is contacted to forward their encoded packets along with the corresponding encoded vectors to the data collector. The data collector builds a *local generating matrix* $\overline{G} \in G$ using the received encoding vectors, an *encoded data matrix* E using the encoded blocks, and solves for the native data B in the system of equations $B = E\overline{G}^{-1}$.

Algebraically, to achieve a survivability s, we attempt to build G such that it is reversible with high probability even when $s \times k$ rows are deleted. According to Theorem 4 (Appendix A), a random binary square matrix achieves its highest probability of reversibility when

$$\frac{\log(k) + h(k)}{k} \geq p \geq 1 - \frac{\log(k) + h(k)}{k} \tag{4}$$

where $h(k)$ is some arbitrary function as in Appendix A. If $h(k)$ equals to some constant c, the resulting invertibility probability (P) can be expressed as

$$c_2 e^{-2e^{-c}} \tag{5}$$

where $c_2 = \pi(0, 2)$ is given by Eq. (A.8). As shown in Fig. 4, it is not difficult to see that $0 \leq P \leq c_2$, for $\mathbb{F}_2$. Further, $P = c_2 \; \forall\, x \geq 7$. The choice of a constant value for $h(k)$ is important since it affects the resulting overhead. Besides, the value of such a constant can be made absolutely small. The following theorem establishes the basis for the design of the proposed DEC-DS.

Theorem 1. *Let G be a $k \times n$ random matrix over $\mathbb{F}_2$, and $s \geq 0$. Further, let $n = (1 + s)k$ and $m = (1 + s)(\log(k) + c_1^\star) + c_2^\star$. G is constructed by selecting m random entries in each of the n columns and setting them to 1. Now, let $\overline{G}$ be a $k \times k'$ matrix constructed by deleting $n - k'$ rows from G chosen uniformly at random, where $k \leq k' \leq n - k$. Then $\overline{G}$ is invertible with a probability $P = c_1 e^{-2e^{-c2}}$ for some constant values $c_1^\star$ and $c_2^\star$.*

Proof. From Theorem 4 (Appendix A), we know that $p = \frac{\log k + h(k)}{k} \leq 1/2$. Also, for a constant $h(k)$, $p = \frac{\log k + c_1^\star}{k}$. Furthermore, Eq. (A.7) shows that on average a constant number of extra packets is required to guarantee invertibility. Let $c_2^\star$ denote the number of extra packets required. Now, to maintain the same invertibility probability for the square $k \times n$ matrix, we need

$$p = \frac{\log(k) + c_1^\star}{k}$$

but,

$$p = \frac{m \times k}{n \times k}$$

hence,

$$m = np = \frac{n}{k}(\log(k) + c_1^\star) + c_2^\star \tag{6}$$

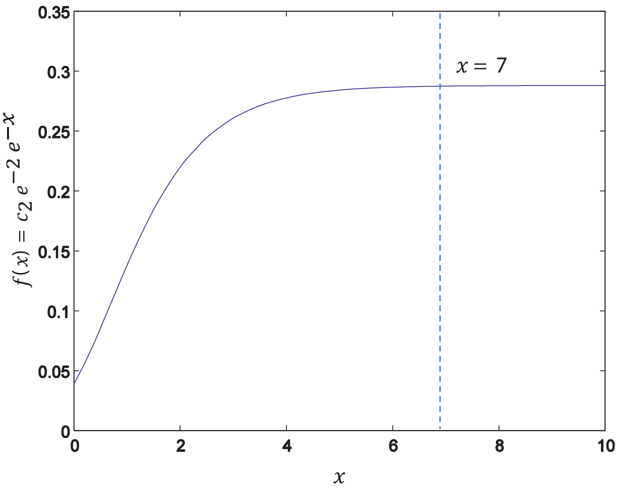


Fig. 4. $P = f(x) = c_2 e^{-2e^{-x}}$.

$$= (1+s)(\log(k) + c_1^\star) + c_2^\star. \tag{7}$$

□

The values of $c_1^\star$ and $c_2^\star$ correspond to the constant c in Eq. (5) and the number of extra vectors required for decoding as in Table A.6, respectively. $c_1^\star$ can be chosen to be ≥ 7 as discussed before. We have also seen that $c_2^\star \geq 8$ guarantees successful decoding with high probability based on the experimental values shown in Table A.6. When data collection takes place, the data collector retrieves e_j's and G_j's to build two matrices E and G, respectively, and decodes the native blocks as $B = EG^{-1}$. The only condition required for this to work is that G must be invertible.

Clearly, there is a compromise between EO and the expected DO. Choosing EO to be small saves energy during dissemination and encoding but results in higher DO during data collection and decoding. Remember that

$$(1+s)(\log k + 7) + 8 \geq m \geq 5(1+s)(\log k). \tag{8}$$

So, the least density of the matrix, and therefore the least energy needed, is achieved when $m = (1+s)(\log(k)+7)+8$. Moreover, the value of $c_2^\star$ is insignificant for large values of k as in Table A.5. In other words, for sufficiently large k, it is sufficient to have $m = (1+s)(\log(k)+7)$.

To summaries, given a network of k source nodes and survivability s, DEC-DS works as follows:

1. Generate n storage nodes where

   $$n = k(s+1). \tag{9}$$

2. Each source node i generates a source data block b_i.
3. Each source node i chooses

   $$m = (s+1)(\log k + c_1^\star) + c_2^\star. \tag{10}$$

   distinct random storage nodes uniformly at random and forwards a copy of b_i to each of the selected nodes.
4. Each storage node combines the received packets linearly.

4. Routing and energy efficiency

By restricting their role to pure routing/forwarding, relay nodes are not fully utilized by existing DEC schemes when disseminating data. So, following generating a source packet, choosing a set of candidate storage nodes, and forwarding the packet by source nodes, relay nodes help forward data packets to storage nodes without manipulating them. However, based on the argument that the cost of communication is generally much higher than processing on wireless nodes, we utilize the coding opportunities that arise during relaying packets using RLNC. Mathematically, since RLNC allows for a broader dissemination of data at a less energy cost, we can achieve the required density of the coefficient matrix using less energy.

The proposed modifications improve the efficiency of the dissemination process by allowing relay nodes to participate in the encoding process during the dissemination phase. This can be done using one of two strategies: Encode-and-Forward (EaF) or Encode-and-Disseminate (EaD). The two schemes that are presented here share the same fundamental model as the DEC-DS scheme we previously presented in Section 3.

4.1. Encode-and-Forward (DEC-EaF)

In each step of the DEC-EaF coding algorithm, a target storage node is randomly chosen by the source node, and the native packet is forwarded accordingly in a multihop fashion. Then, for every relay node by which the packet passes, the relay node combines the packet it receives with the encoded packets stored locally before forwarding the new packet to the next hop. If no packet exists locally, the relay node simply saves a copy of the relayed packet. The source node is assumed to have multiple routes for every destination node. While the choice of the destination node is random, selecting the best route is not.

Table 2 summaries all the notations that will be used in the description of the proposed schemes. Let $R_j = \{r_1, r_2, \ldots, r_h\}$ be the set of possible routes to a

Table 2
Notations used in DEC-EaF and DEC-EaD.

Attribute	Meaning
$\mathbb{R}_i$	Possible routes to node y_i
r_l	lth Route to in R_j
s_w	A node in the route r_l
Z	Nodes previously visited
$\overline{Z}$	Nodes not yet visited
x_i	Source node i
$\overline{m}$	Current redundancy factor
σ	Depletion of route i
ω	Route selection gain
$\overline{r}$	Average hop count
θ	Remaining copies to be disseminated
ω_{index}	next hop for random walk

destination s_j. Further, let $r_l = \{s_1, s_2, \ldots, s_w\}$ be the set of nodes in route r_l. Let Z be the set of nodes that have been either chosen as destination nodes or those which were in routes to previously selected destination nodes. In other words, Z is the set of visited nodes. On the other hand, $\overline{Z}$ is a set of nodes that have not yet been visited. Every time a destination s_i and a route r_l pair are selected, Z is updated as follows

$$Z = Z \cup s_i \cup r_l. \tag{11}$$

We define the current redundancy factor $\overline{m}$ as a parameter to track the number of copies that have been encoded on storage nodes. Accordingly, the current redundancy factor $\overline{m}$ is updated as

$$\overline{m} = m - |Z|. \tag{12}$$

In addition to subtracting the number of visited nodes from $\overline{m}$, the corresponding nodes that have been visited are also removed from candidate destination nodes as follows

$$\overline{Z} = \overline{Z} - (\overline{Z} \cap Z). \tag{13}$$

We also define the depletion (σ) of a route r_1 as

$$\sigma(r_1) = |r_1 \cap Z|. \tag{14}$$

Furthermore, we define the route selection gain (ω) of a route r_1 as

$$\omega(r_1) = |r_1| - \sigma. \tag{15}$$

Next, suppose s_i is randomly selected as a destination. Amongst the possible routes $(\mathbb{R}_i)$ to s_i we select r_i such that

$$\{r_i | \omega(r_i) > \omega(r)\} \ \forall r \in \mathbb{R}_i. \tag{16}$$

Equivalently, DEC-EaF chooses the shortest path routes with the least number of visited nodes.

Algorithms 1 and 2 show pseudo codes describing the mechanism at the source node and the relay nodes, respectively.

It should be noted that the performance of the proposed scheme depends on the topology of the network and the routing protocol in use. Let r_{ij} be the hop count between source node x_i and storage node y_j. Let $R = \{r_{ij}\} \ \forall i \in X, j \in Y$ be a $k \times n$ matrix containing the hop count between every pair of source and storage nodes. The average

Algorithm 1 DEC Encode-and-Forward [DEC-EaF] (Source Node).

Generate a packet x_i
$\overline{m} \leftarrow m$
$Z \leftarrow \phi$
Calculate σ for each r_i
for $j = 1 \to m$ AND $\overline{m} > 0$ **do**
  Selects a target storage node s_j uniformly at random
  Choose r according to Eq. (16)
  Route x_i to s_j through r
  $Z = Z \cup s_i \cup r$
  $\overline{m} = m - |Z|$
  $\overline{Z} = \overline{Z} - (\overline{Z} \cap Z)$.
end for

Algorithm 2 DEC Encode-and-Forward [DEC-EaF] (Relay Node).

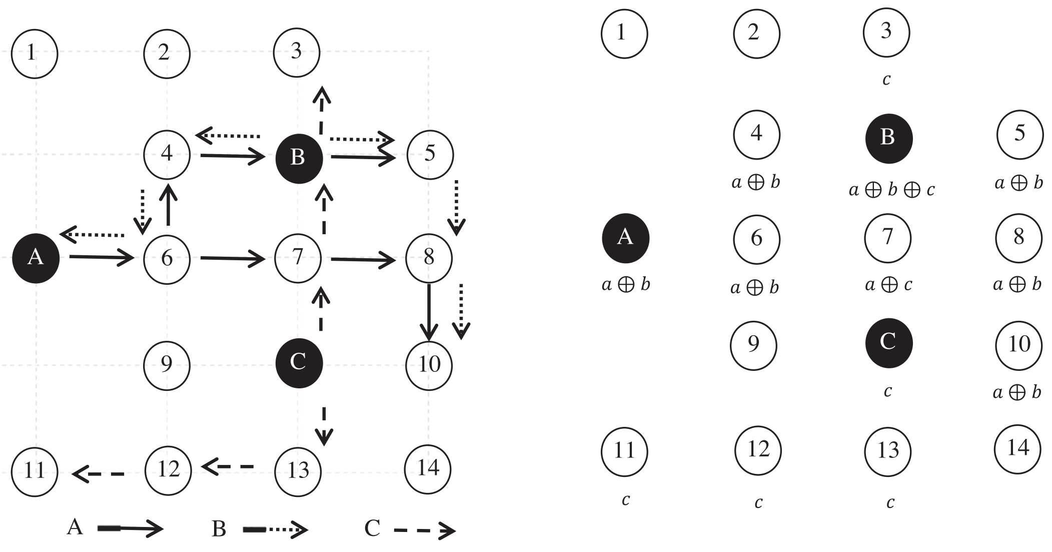
Receive packet x_i at *relay node*
if $g_i = 0$ **then**
  Generate a new coefficient g_{ji}
  $e_j = e_j \oplus (g_{ji} \times x_i)$
  $g_i = g_{ji}$
end if
Forward x_i to next node in route;

hop count $\overline{r}$ can be calculated as

$$\overline{r} = \frac{\sum_{i=1}^{k} \sum_{j=1}^{n} r_{ij}}{k \times n}. \tag{17}$$

The expected reduction in the required redundancy factor m should be roughly $\overline{r}$. However, due to the fact that the routes between nodes are not all disjoint, some relay nodes may be visited by the same packets multiple times.

An example of how DEC-EaF works is shown in Fig. 5. The nodes A, B, and C are assumed to be the source nodes while the nodes denoted by numerals are the storage nodes. First, node A chooses node 10 as a destination storage node. Clearly, the shortest paths between A and 10 are 3 hops long, which are {6, 7, 8, 10} and {6, 7, C, 10}. We assume {6, 7, 8, 10} is chosen randomly to break the tie. Since 5 copies of the source data have been disseminated during the last step (including the source node A), the new $\overline{m} = 7 - 5 = 2$. In the next step, assume node 5 has been selected. Two candidate shortest path routes exist between A and 5; which are {6, 7, 8, 5} and {6, 4, B, 5}. However, the number of unvisited nodes in the former equals 1 while in the latter equals 3. Therefore, {6, 4, B, 5} is chosen. If both routes have the same number of unvisited nodes, one is selected randomly. Now, since the $\overline{m}$ becomes < 1, the dissemination process stops. The same logic can be applied for node B, using node 10 and route {5, 8, 10} and node A and route {4, 6, A}. Finally, node C selects and reaches node 11 through {13, 12, 11} and node 3 through {7, B, 3}. The resulting code is shown in Fig. 5(b).



(a) Dissemination (b) Encoding

Fig. 5. Example of DEC-EaF: $m = 7$.

4.2. Encode-and-Disseminate (DEC-EaD)

In the second strategy, called DEC-EaD, source nodes disseminate the source packets using a node-centric random walk mechanism. Since the selection of target nodes is random in the original DEC-DS, it makes sense to use a random walk to eliminate the need for routing table construction and maintenance. In random walk protocols, there is no guarantee that a certain packet will not visit the same node more than once.

The relay forwarding strategy in DEC-EaD is based on the rotor-router model which is a quasirandom analog to the random walk process. The rotor-router model has been introduced in [34] and popularized by Jim Propp in 2001, and has attracted a lot of interest. The main advantage of the model in our application is that it reduces the chance of forwarding packets to the same node when there are some neighbors that have not been contacted. In addition, the model is simple and eliminates the need for maintaining a forwarding table for each source packet.

In DEC-EaD, each node maintains a list of all single-hop neighbors. In addition, each relay node maintains an index for each packet. The index points to the next neighbor to whom each packet will be forwarded when the packet passes by the current relay node. Whenever a packet is forwarded, the corresponding index is advanced to the following neighbor. This is required to reduce chances of some packets revisiting the same nodes. Note that the order in which neighboring nodes are selected is immaterial.

Like in DEC-EaF, when a relay node receives a packet, it encodes it locally before forwarding. Then, it forwards the packet to one neighbor according to the corresponding index. To track the distribution of the coding process, each packet contains a redundancy factor counter (θ) which tracks the number of copies of the packet remaining to be disseminated. The counter is decreased by one at each newly visited hop. When the counter reaches zero, the random walk terminates. The process is formally described in Algorithms 3 and 4.

Algorithm 3 DEC Encode-and-Disseminate [DEC-EaD] (Source Node).

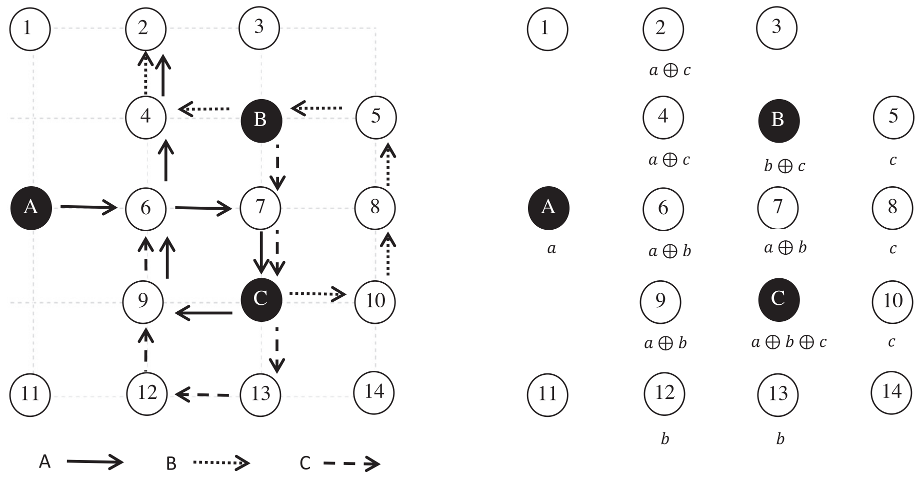
Discover neighboring nodes
Generate a packet x_i
$\theta \leftarrow m$
Choose a Neighbor w_{index}
Forward (x_i, θ) to w_{index}
Advance *index*

Algorithm 4 DEC Encode-and-Disseminate [DEC-EaD] (Relay Node).

Receive Packet x_i;
if $g_i = 0$ **then**
  Generate a new coefficient g_i;
  $e_j = e_j \oplus (g_i \times x)$;
end if
$\theta = \theta - 1$;
if $\theta > 0$ **then**
  Forward x_i to node $w_{(index)}$;
  Advance *index*
end if

Note that DEC-EaD does not require any routing. All what is required is knowing the set of neighbors using a neighbor-discovery mechanism. This is clearly an added advantage over DEC-EaF besides the reduction in energy as will be shown in the next section.

An example of the operation of DEC-EaD is shown in Fig. 6 for $m = 7$. Node A starts the following random walk {6, 7, C, 9, 6, 4, 2}. At each hop, the relay node chooses one neighbor (different from the last hop) uniformly at



(a) Dissemination (b) Encoding

Fig. 6. An Example of DEC-EaD: $m = 7$.

random. In addition, each relay node decrements the value of $\overline{m}$ in the packet by 1. When the random walk reaches node 6 for the second time, it is forwarded to node 4, since all other neighbors have been visited. In such a case, $\overline{m}$ remains unchanged. Finally, the random walk terminates at node 2. Similarly, node B and C executes their random walks as {7, C, 13, 12, 9, 6} and {10, 8, 5, B, 4, 2}. The resulting code is shown in Fig. 6(b).

5. Performance evaluation

As we stated earlier, our proposed schemes target resource limited networks. Therefore, they must achieve data survivability at a reasonable energy cost. We also conjectured that data survivability can be achieved using less redundancy than required by the original DEC. In this section we show through experimentation that DEC-DS can guarantee data survivability while reducing redundancy requirements. Note that redundancy reduction impacts the energy required for data dissemination and encoding. We also show that using RLNC results in remarkable energy savings while achieving data survivability.

To evaluate the performance of the proposed schemes, the following experimental setup is developed and tested using simulation. The simulator takes as input: k, s, and $\mathbb{F}_q$, in addition to the required dissemination mechanism. Since this study targets applications for WSNs, we use $\mathbb{F}_2$ in our experiments. Based on the number of source nodes (k) supplied and the survivability (s) required, the values for n and m are calculated using Eqs. (9) and (10). Based on the justification given previously, we set $c_1^\star = 7$ and $c_2^\star = 8$.

The simulator starts by creating a network of k source nodes and a storage network of n nodes. Note that in a real implementation source nodes can also serve as storage nodes. However, to simplify our simulation and analysis, we assume no overlap between the sets of source and storage nodes. Next, the dissemination and encoding phase begins using the dissemination mechanism of choice, i.e., DEC, DEC-DS, DEC-EaF, or DEC-EaD. After the code is build, the survivability of the code is tested for values corresponding to survivability from 1 to s. To test the code survivability, we set f, the erasure rate, to values between 0 and 1. For each value of f, $f \times n$ uniformly randomly selected storage nodes are deleted. Then, data collection is carried out, and decoding is executed to test if the k native data packets can still be decoded. To test decodability, storage nodes are selected randomly to build $\overline{G}$. If $rank(\overline{G}) = k$, decoding is successful and we record the number of packets used for decoding (β). Otherwise, decoding fails. In addition, the number of transmit, receive, and processing operations executed during dissemination and encoding are recorded.

Algorithm 5 Simulation Pseudo-code.

Require: k, s, and $\mathbb{F}_2$
  $n = (s + 1)k$;
  $m = (s + 1)(\log k + 7) + 8$;
for $i = 1$ to 1000 **do**
  Disseminate(n, m);
  for failure level (f) from 0 to 1 **do**
    CollectPackets(f);
  end for
end for

To calculate the probability of successful decoding (P_s), large number of different test cases are generated and tested. Algorithm 5 illustrates the general steps of the simulation.

The simulation was carried out for $k = 10, 20, 30, 40$, and 50. The results are based on 1000 different runs using different initial random choices of storage nodes by source nodes. In the first part of the evaluation we are interested in the coding-related aspects of the proposed schemes, namely, probability of successful decoding (P_s) and the average number of packets required for

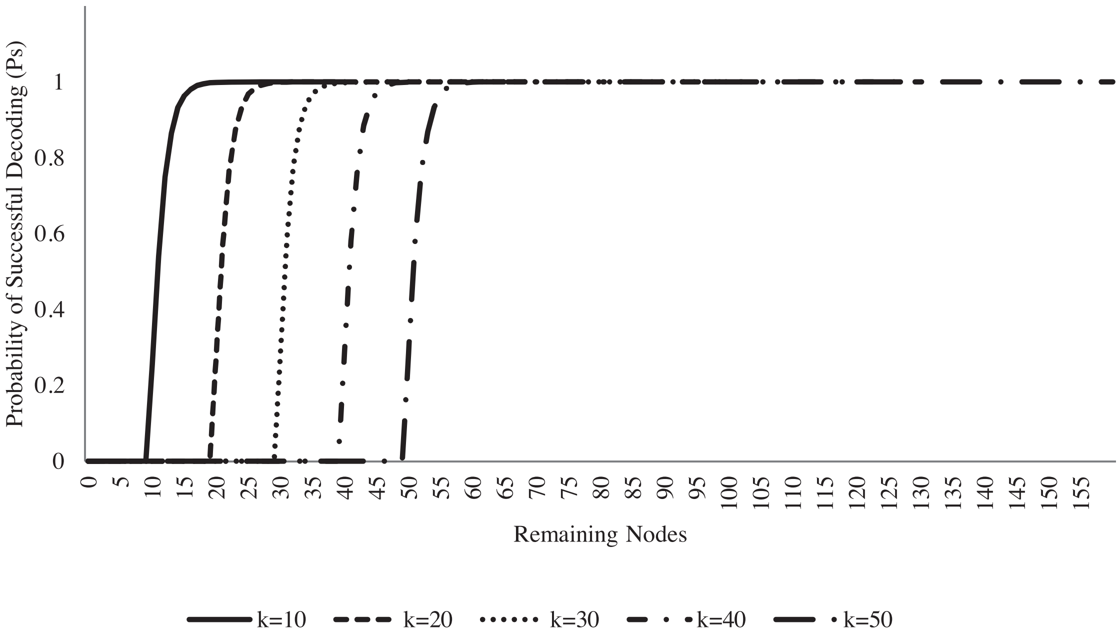


Fig. 7. Probability of successful decoding (P_s): $k = 10 - 50$, $s = 2$, $\mathbb{F}_2$.

decoding (β). The performance of the three schemes is similar, therefore, we presented the results without designation of the used scheme. P_s can be defined as

$$P_s = \frac{\text{Number of successful decodings}}{\text{Total number of trials}}. \tag{18}$$

Fig. 7 shows the performance of the resulting code in terms of P_s. The graph shows the values of P_s for different values of k and for $s = 2$. The probability are as expected since $P_s = 1 \ \forall f \leq \frac{s}{s+1} = 0.67$. When $f > 0.67$, no enough encoded data exist to recover all the native data. It is also interesting to note that the code works even for networks as small as $k = 10$. The decoding overhead (β) represents the number of packets needed to successfully complete the decoding process. As shown in Fig. 8, only one or two packets are required on average beyond k to successfully decode all packets. This is true for all values of k as small as 10. We are also interested in the redundancy factor (m) to compare the proposed scheme with the scheme in [6]. As shown in Fig. 9, DEC-DS requires less redundancy than the original DEC to achieve data survivability. Furthermore, the lowering redundancy does not compromise the decodability of the code. The savings in redundancy translate to lower energy requirements when applied to resource limited systems such as WSNs.

Beside the performance of the code with regard to survivability, it is essential to examine the energy needed to implement each scheme. In case of DEC-DS and DEC-EaF, a multipath shortest routing table is generated beforehand. In addition, both algorithms use a link-state routing strategy. Before, the dissemination phase starts, the routing table is built with multiple routes for each destination in case of DEC-EaF. On the other hand, DEC-EaD does not require any routing table; and neighbor discovery takes place instead. Although the network may become disconnected during data collection phase due to failure of nodes, we assume the data collector is powerful enough to reach all nodes. This assumption is driven by our focus on "data survivability" rather than "network survivability".

In order to quantify the energy requirements, we define the following energy model. We assume a network composed of wireless nodes powered by batteries. The model accounts for the energy consumed due to encoding, transmission, relaying, and reception. The focus of this work is on the number of operations executed during the dissemination and encoding process. Hence, we use an abstract energy model as described here.

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

Accordingly, the cost of forwarding a packet by a relay node without using RLNC is $\xi_F = \xi_r + \xi_t$ for one reception and one transmission. On the other hand, the cost of relaying with RLNC becomes $\xi_{RLNC} = \xi_r + \xi_e + \xi_t$. The extra cost accounts for generating random coefficients, multiplying the received packet by the random coefficient, and *xor*-ing the result with the local encoded packet. Also, we do not consider the energy consumed by sensing since it is assumed to be equal for all nodes and can be factored out.

Table 4 shows the energy required to implement each scheme for different values of k. We can see that DEC-EaF and DEC-EaD can implement the required code using remarkably less energy compared to DEC-DS. The number of send (SND) and receive (REC) operations are equal for each scheme since each single "send" operation implies a single "receive" operation. It is generally the case in WSNs that send and receive operations consume more energy than

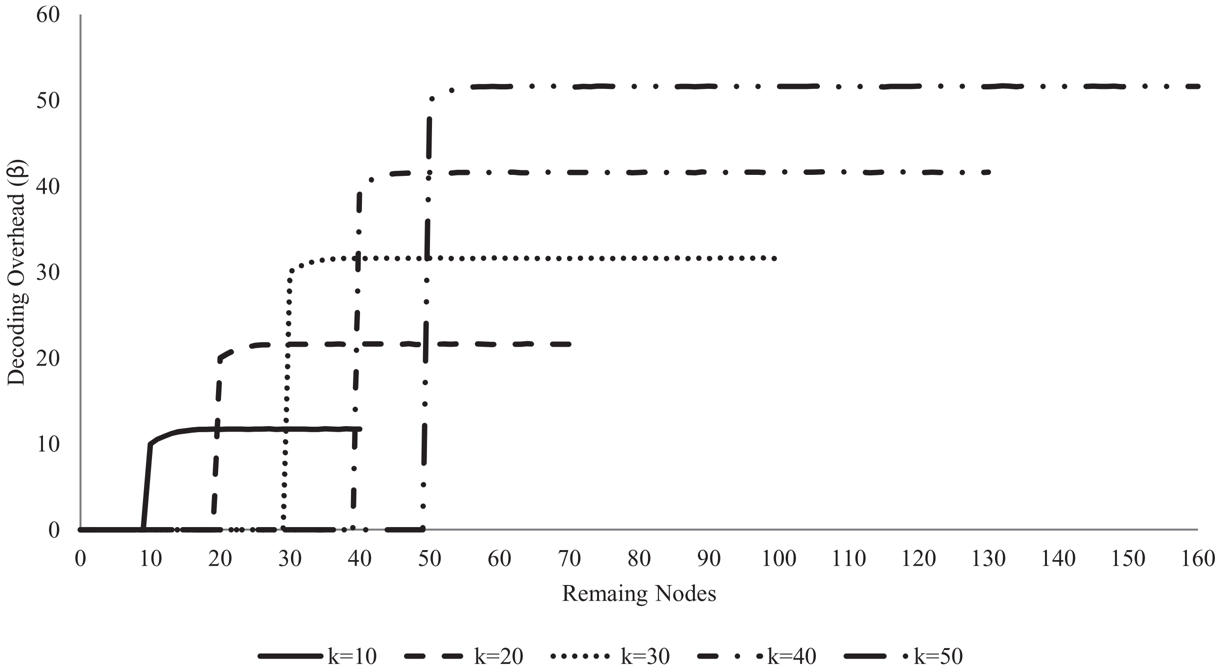


Fig. 8. Decoding overhead (β): $k = 10 - 50$, $s = 2$, $\mathbb{F}_2$.

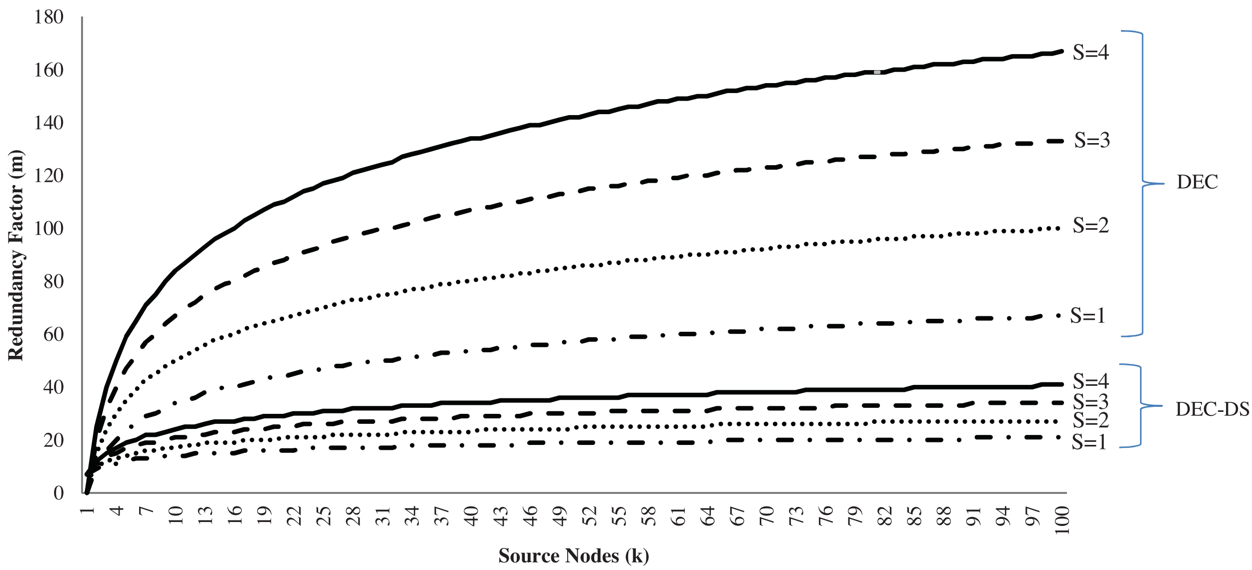


Fig. 9. Redundancy factor (m) as a function of source nodes (k).

local processing. When taking this into consideration, we see that the local encoding performed by RLNC not only contributes to the total energy consumed by the scheme, it saves a significant amount of energy over all. It can also be seen that DEC-DS and DEC-EaD achieve the exact number of encodings while DEC-EaF does not. This is because when the algorithm reaches the required *RF*, the number of unvisited nodes on the selected route may be more than what is required, which results in the extra encodings. This can be easily enhanced, especially if the number of nodes is large, the cumulative effect could be significant.

To express the performance in terms of energy, we compile the number of operations into energy figures using the energy requirements of the CC1000 chip. On the CC1000 chip running at 868 MHz, processing requires 5 mW while “send” and “receive” consumes 25.8 mW and 28.8 mW, respectively, as shown in Table 3 [35]. Therefore, coding for a $k = 20$ network requires on the same chip needs 4.5 W for DEC, 3 W for both the DEC-DS and DEC-EaD, and 3.11 W for the DEC-EaF. On the other hand, communications require 237.6 W, 126.7W, 24.7 W, and 50.5 W for DEC, DEC-DS, DEC-EaF, and DEC-EaD, respectively. Even

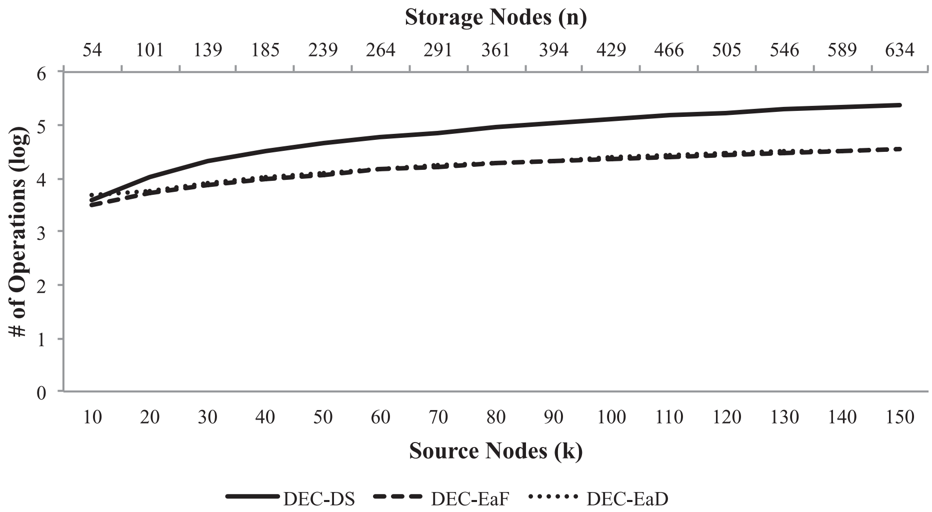


Fig. 10. Energy vs. network size for $s = 4$.

Table 3
Power requirements of the CC1000 chip running at 868 MHz.

CC1000			
Power (mW)	Process (ξ_e) 5	Transmit (ξ_t) 25	Receive (ξ_r) 28.8

though the performance of the DEC-EaF is superior to that of DEC-EaD, it comes at the cost of energy needed for routing. Given the difference between the two, DEC-EaF may be more feasible for those WSN applications where routing is not required.

Fig. 10 shows how the energy requirements increase as a function of the network size. Note that the *Y*-axis is in a logarithmic scale. It is evident that the savings achieved by both DEC-EaF and DEC-EaD are substantial compared to DEC-DS. Also, there does not seem to be a considerable difference in energy requirements between DEC-EaF and DEC-EaD. Note that we observed the same trend for higher values of *s* as illustrated in Fig. 10.

6. Conclusion

In this paper, we introduce a DDSS for data survivability in WSNs based on DEC. The framework aims at determining the amount of redundancy in both storage and data, and the maximum level of failure that can be tolerated without losing in-network data. Compared to the original DEC which aim at reducing DO, the proposed schemes

Table 4
Number of coding, send, receive operations, and total energy required to implement data survivability schemes.

k	*n*	Operation	DEC	DEC-DS	DEC-EaF	DEC-EaD
10	54	COD	350	280	286.357	280
		SND	1174.82	847.375	421.127	496.978
		REC	1174.82	847.375	421.127	496.978
Total power (W)			64.60	47.67	28.54	24.43
20	101	COD	900	600	622.24	600
		SND	4440.67	2321.6	857.448	925.928
		REC	4440.67	2321.6	857.448	925.928
Total power (W)			242.08	129.76	53.56	49.93
30	139	COD	1560	960	1003.38	960
		SND	9559.87	4393.854	1356.294	1441.42
		REC	9559.87	4393.854	1356.294	1441.42
Total power (W)			519.25	244.70	83.50	79.07
40	185	COD	2240	1320	1393.7	1320
		SND	15561.31	6869.34	1811.988	1926.99
		REC	15561.31	6869.34	1811.988	1926.99
Total power (W)			843.73	381.7	111.81	105.90
50	239	COD	2950	1650	1759.714	1650
		SND	22877.79	9563.376	2216.028	2354.948
		REC	22877.79	9563.376	2216.028	2354.948
Total power (W)			1238.71	530.41	136.83	129.79

target achieving data survivability. Due to the random nature of the DEC-DS scheme, it can be implemented in a decentralized manner without coordination between the sensor nodes involved. Since the framework is targeting WSNs applications, we show two schemes utilizing RLNC to reduce the energy needed to implement the storage system. Even though the scheme is discussed under the assumption of a WSN, we believe the scheme is general enough to be also applied to other network architectures. An interesting dimension to pursue is to study the data update problem where outdated data need to be replaced by new ones. Another direction is incorporating Quality of Service (QoS) measures into the framework. In other words, extending the framework to situations where different classes of data demand different data survivability requirements. We plan to consider this interesting problem in a future study.

Appendix A. Rank properties for random matrices over finite field

We review some important results on the properties of random matrices over $\mathbb{F}_2$. Interested readers are directed to [36] for a detailed discussion. Let G be a $k \times m$ random matrix where each element g_{ij} of G is drawn from $\mathbb{F}_2$ according to a probability distribution $\rho(x)$. More specifically,

$$\rho(x) = P(g_{ij} = x) = \begin{cases} 1-p, & \text{for } x = 0 \\ p & \text{otherwise .} \end{cases} \tag{A.1}$$

We are interested in the probability of G having a certain rank r, in terms of the column count m. It would be useful to note that the number of possible vectors in the k-dimensional space $\mathbb{F}_q^k$ is q^k. In the case where $q = 2$ and $p = 1/2$ (uniform distribution), the probability of $G_{k\times m}$ having a rank $r = m$, can be expressed as

$$P(rank(G) = m) = (1-2^{-k})(1-2^{-(k-1)})\dots(1-2^{-(k-m+1)}) \tag{A.2}$$

$$= \prod_{i=k-m+1}^{k} (1-2^{-i}). \tag{A.3}$$

Specifically, when $m = k$, the probability of G being full rank is

$$P(rank(G) = k) = \prod_{i=1}^{k} (1-2^{-i}), \tag{A.4}$$

where the first term in Eq. (A.2) represents the probability of choosing the first vector, namely, choosing any vector except the zero vector ($\vec{0}$). The second term represents the probability of choosing any vector except any linearly dependent vector of the vector chosen in the previous step and $\vec{0}$. The third term corresponds to choosing any vector other than the already chosen vector or any of their linear combinations. The rest of the formula can be deduced similarly.

Interestingly, Eq. (A.2) converges to a constant when $k \to \infty$. To see this, consider the following theorem from [37].

Theorem 2. *Let G be a binary random $k \times n$, $n \geq 0$ matrix with entries chosen equally likely. Then for $k - s \leq min(k, n)$, $k \leq s \leq 0$ and $k \to \infty$ we have*

$$P(rank(G) = k-s) \to 2^{-s(m+s)} \prod_{i=s+1}^{\infty} (1-\frac{1}{2^i}) \prod_{i=1}^{m+s} (1-\frac{1}{2^i})^{-1},$$

where the last product equals 1 for $m + s = 0$ (i.e. full rank matrix).

Now, let Q_s denote the probability that a $k \times m$ matrix has a rank $r = min(k, m)$. Then

$$Q_s = \prod_{i=s+1}^{\infty} (1-2^{-i})$$

$$\log(Q_s) = \log \prod_{i=s+1}^{\infty} (1-2^{-i}) = \sum_{i=s+1}^{\infty} \log(1-2^{-i})$$

$$= \sum_{i=1}^{\infty} \frac{-2^{si}}{i(2^i-1)}.$$

Q_0 can be seen as the probability that the matrix has a full rank given that k columns have been generated. It can be computed as

$$Q_0 = \prod_{j=1}^{\infty} \left(1-\frac{1}{2^i}\right) = 0.2887880951\dots \tag{A.5}$$

Now, let P_m be the probability that exactly m extra packets beyond k are needed to achieve full rank. P_m can be expressed as

$$P_m = Q_m - Q_{m-1}. \tag{A.6}$$

Therefore, the average number of extra packets $\overline{m}$ required to achieve full rank equals

$$\overline{m} = \sum_{m=0}^{\infty} mP_m = \sum_{i=0}^{\infty} (1-Q_i) = 1.6067. \tag{A.7}$$

There are two important results to note from Eqs. (A.5) and (A.7). First, the probability of full rank of any uniformly distributed square matrix converges to a constant. This is in fact true to binary as well as non-binary matrices. In the case of binary matrices, this constant is 0.288. Table A.5 shows Q_0 for different values of q. Second and more importantly, the number of extra vectors (beyond k) required to have a full rank with high probability is on average very low and is independent of k. For example, on average only two extra vectors are required to make G have a full rank. As shown on Table A.6, when the number of extra vectors is 8, the probability is $Q_0 = 0.996$. The latter is quite accurate for k as low as 10.

The following two important theorems state that the full rank probability seen above is not specific to the uniform distribution ($p = \frac{1}{2}$). In fact, as long as the probability p is within a certain interval, the results from the uniform case still apply. To see this, consider Fig. A.11. The plot was generated for a randomly generated square matrix with $k = 20$ and it shows the average invertibility probability versus p for $\mathbb{F}_2$. The curve illustrates that the invertibility probability of 0.288 applies for a range of values of p and not only for $p = 1/2$.

In [38,39], Cooper shows an expression of the probability of the rank of a random matrix over a finite field in

Table A.5
Invertibility probability vs. finite field (F_q).

q	2	4	8	16	32	64	128	256
Q_0	0.288	0.689	0.859	0.934	0.968	0.984	0.992	0.996

Table A.6
Invertibility probability vs. number of extra vectors ($\beta - k$).

s	0	1	2	3	4	5	6	7	8
Q_s	0.288	0.577	0.770	0.880	0.938	0.969	0.985	0.992	0.996

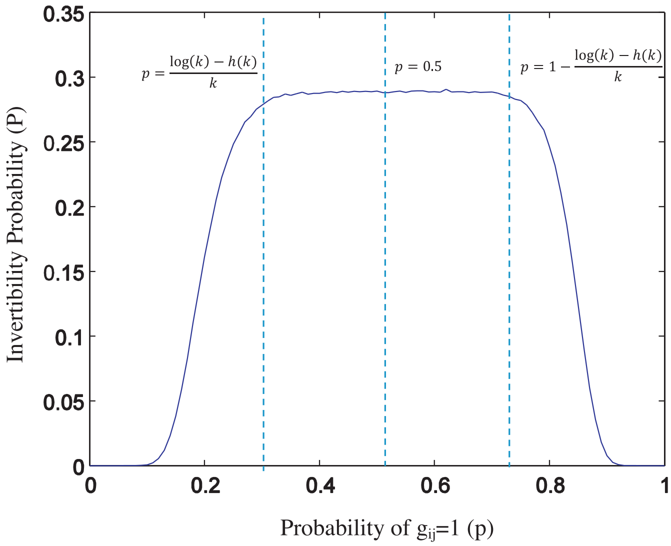


Fig. A.11. Invertability probability (P) vs. p for $\mathbb{F}_2$.

terms of p. The expression applies to p over certain period according to the following theorem.

Theorem 3 ([38], Theorem 1)**.** *Let $p = \frac{q-1}{q}$, and let G be a random $(k \times k)$-matrix with entries in $\mathbb{F}_q$. Let $p_n(s, q)$ be the probability that $rank(G) = k - s$, then*

$$\lim_{k\to\infty} p_n(s,q) = \pi(s,q) \tag{A.8}$$

$$= \begin{cases} \prod_{j=1}^{\infty}(1-\frac{1}{q^j}), & s = 0 \\ \frac{\prod_{j=s+1}^{\infty}(1-\frac{1}{q^j})}{\prod_{j=1}^{s}(1-\frac{1}{q^j})}(\frac{1}{q})^{s^2}, & s \geq 1. \end{cases} \tag{A.9}$$

Theorem 4 ([38], Theorem 2-i)**.** *Let $G \in G(k, p, 2)$ be a $k \times k$ random binary matrix over $\mathbb{F}_2$. Further, if $p(k) = \frac{\log k + h(k)}{k} \leq 1/2$: Then*

$$\lim_{k\to\infty} P(G \text{ is non-singular}) = \begin{cases} 0, & h(n) \to -\infty \\ c_2 e^{-2e^{-h}}, & \text{h constant} \\ c_2, & h(n) \to \infty \end{cases} \tag{A.10}$$

where $c_2 = \pi(0, 2)$.

References

[1] W. Wu, J. Zhang, A. Luo, J. Cao, Distributed mutual exclusion algorithms for intersection traffic control, IEEE Trans. Parallel Distrib. Syst. 26 (1) (2015) 65–74, doi:10.1109/TPDS.2013.2297097.
[2] S. Bera, S. Misra, J. Rodrigues, Cloud computing applications for smart grid: A survey, IEEE Trans. Parallel Distrib. Syst. 26 (5) (2015) 1477–1494, doi:10.1109/TPDS.2014.2321378.
[3] H. Ning, H. Liu, L. Yang, Aggregated-proof based hierarchical authentication scheme for the Internet of Things, IEEE Trans. Parallel Distrib. Syst. 26 (3) (2015) 657–667, doi:10.1109/TPDS.2014.2311791.
[4] Z. Ren, M. Hail, H. Hellbruck, CCN-WSN - A lightweight, flexible content-centric networking protocol for wireless sensor networks, in: Proceedings of the 2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, 2013, pp. 123–128, doi:10.1109/ISSNIP.2013.6529776.
[5] A.G. Dimakis, V. Prabhakaran, K. Ramchandran, Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes, in: Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks (IPSN '05), Piscataway, NJ, USA, 2005, p. 15.
[6] A. Dimakis, V. Prabhakaran, K. Ramchandran, Distributed data storage in sensor networks using decentralized erasure codes, in: Proceedings of the Thirty-eighth Asilomar Conference on Signals, Systems, and Computers, vol. 2, 2004, pp. 1387–1391, doi:10.1109/ACSSC.2004.1399381.
[7] A. Dimakis, V. Prabhakaran, K. Ramchandran, Decentralized erasure codes for distributed networked storage, IEEE Trans. Inf. Theory 52 (6) (2006) 2809–2816, doi:10.1109/TIT.2006.874535.
[8] R. Ahlswede, N. Cai, S.-Y. R. Li, R.W. Yeung, Network information flow, IEEE Trans. Inf. Theory 46 (4) (2000) 1204–1216.
[9] C. Fragouli, J.-Y. Le Boudec, J. Widmer, Network coding: An instant primer, ACM SIGCOMM Comput. Commun. Rev. 36 (1) (2006) 63–68, doi:10.1145/1111322.1111337.

[10] S.-Y. R. Li, R.W. Yeung, N. Cai, Linear network coding, IEEE Trans. Inf. Theory 49 (2) (2003) 371–381.
[11] H. Weatherspoon, J. Kubiatowicz, Erasure coding vs. replication: A quantitative comparison, in: Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '01), Springer-Verlag, London, UK, 2002, pp. 328–338.
[12] S. Acedan'ski, S. Deb, M. Médard, R. Koetter, How good is random linear coding based distributed networked storage, in: Proceedings of the First Workshop on Network Coding, Theory, and Applications (NetCod 2005), 2005.
[13] F.A. Kuipers, An overview of algorithms for network survivability, ISRN Commun. Netw. 2012 (2012) 19. Article ID 932456.
[14] J. Xu, J. Tang, K. Kwiat, W. Zhang, G. Xue, Enhancing survivability in virtualized data centers: A service-aware approach, IEEE J. Sel. Areas Commun. 31 (12) (2013) 2610–2619, doi:10.1109/JSAC.2013.131203.
[15] J.W. Byers, M. Luby, M. Mitzenmacher, A. Rege, A digital fountain approach to reliable distribution of bulk data, in: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '98), ACM, New York, NY, USA, 1998, pp. 56–67, doi:10.1145/285237.285258.
[16] I.S. Reed, G. Solomon, Polynomial codes over certain finite fields, J. Soc. Ind. Appl. Math. 8 (2) (1960) 300–304, doi:10.1137/0108018.
[17] M. Luby, LT codes, in: Proceedings of the Forty-third IEEE Annual Symposium on Foundations of Computer Science, 2002, pp. 271–280, doi:10.1109/SFCS.2002.1181950.
[18] A. Shokrollahi, Raptor codes, IEEE Trans. Inf. Theory 52 (6) (2006) 2551–2567, doi:10.1109/TIT.2006.874390.
[19] D.J.C. Mackay, Fountain codes, IEE Commun. 152 (2005) 1062–1068.
[20] A. Dimakis, V. Prabhakaran, K. Ramchandran, Distributed fountain codes for networked storage, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006), vol. 5, 2006, p. V, doi:10.1109/ICASSP.2006.1661484.
[21] Y. Lin, B. Liang, B. Li, Data persistence in large-scale sensor networks with decentralized fountain codes, in: Proceedings of the Twenty-sixth IEEE International Conference on Computer Communications (INFOCOM), 2007, pp. 1658–1666, doi:10.1109/INFCOM.2007.194.
[22] S.A. Aly, Z. Kong, E. Soljanin, Fountain codes based distributed storage algorithms for large-scale wireless sensor networks, in: Proceedings of the Seventh International Conference on Information Processing in Sensor Networks (IPSN '08), IEEE Computer Society, Washington, DC, USA, 2008, pp. 171–182, doi:10.1109/IPSN.2008.64.
[23] S. Aly, Z. Kong, E. Soljanin, Raptor codes based distributed storage algorithms for wireless sensor networks, in: Proceedings of the IEEE International Symposium on Information Theory (ISIT 2008), 2008, pp. 2051–2055, doi:10.1109/ISIT.2008.4595350.
[24] Z. Kong, S. Aly, E. Soljanin, Decentralized coding algorithms for distributed storage in wireless sensor networks, IEEE J. Sel. Areas Commun. 28 (2) (2010) 261–267, doi:10.1109/JSAC.2010.100215.
[25] D. Vukobratovic, C. Stefanović, M. Stojakovic, V. Stanković, Raptor packets: A packet-centric approach to distributed raptor code design, in: Proceedings of the IEEE International Symposium on Information Theory (ISIT 2009), 2009, pp. 2336–2340, doi:10.1109/ISIT.2009.5205950.
[26] D. Vukobratovic, C. Stefanović, V. Crnojević, F. Chiti, R. Fantacci, A packet-centric approach to distributed rateless coding in wireless sensor networks, in: Proceedings of the Sixth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '09, 2009, pp. 1–8, doi:10.1109/SAHCN.2009.5168905.
[27] D. Vukobratovic, C. Stefanović, V. Stanković, Fireworks: A random linear coding scheme for distributed storage in wireless sensor networks, in: Proceedings of the 2010 IEEE Information Theory Workshop (ITW), 2010, pp. 1–5, doi:10.1109/CIG.2010.5592800.
[28] S. Kokalj-Filipovic, P. Spasojevic, E. Soljanin, Doped fountain coding for minimum delay data collection in circular networks, IEEE J. Sel. Areas Commun. 27 (5) (2009) 673–684, doi:10.1109/JSAC.2009.090609.
[29] 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.
[30] Q. Wang, K. Ren, S. Yu, W. Lou, Dependable and secure sensor data storage with dynamic integrity assurance, ACM Trans. Sen. Netw. 8 (1) (2011) 9:1–9:24, doi:10.1145/1993042.1993051.
[31] R. Zeng, Y. Jiang, C. Lin, Y. Fan, X. Shen, A distributed fault/intrusion-tolerant sensor data storage scheme based on network coding and homomorphic finger printing, IEEE Trans. Parallel Distrib. Syst. 23 (10) (2012) 1819–1830, doi:10.1109/TPDS.2011.294.
[32] M.A. Mahmood, W.K. Seah, I. Welch, Reliability in wireless sensor networks: A survey and challenges ahead, Comput. Netw. 79 (2015) 166–187, doi:10.1016/j.comnet.2014.12.016.
[33] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, J. Crowcroft, Xors in the air: Practical wireless network coding, IEEE/ACM Trans. Netw. 16 (3) (2008) 497–510, doi:10.1109/TNET.2008.923722.
[34] V.B. Priezzhev, D. Dhar, A. Dhar, S. Krishnamurthy, Eulerian walkers as a model of self-organized criticality, Phys. Rev. Lett. 77 (1996) 5079–5082, doi:10.1103/PhysRevLett.77.5079.
[35] Q. Wang, M. Hempstead, W. Yang, A realistic power consumption model for wireless sensor network devices, in: Proceedings of the Third Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON '06), vol. 1, 2006, pp. 286–295, doi:10.1109/SAHCN.2006.288433.
[36] C. Studholme, I.F. Blake, Properties of random matrices and applications, Unpublished Report, http://www.cs.toronto.edu/~cvs/coding.
[37] V.F. Kolchin, Random Graphs, Cambridge University Press, New York, NY, USA, 1999.
[38] C. Cooper, On the rank of random matrices, Random Struct. Algorithms 16 (2000) 209–232, doi:10.1002/(SICI)1098-2418(200003)16:2⟨209::AID-RSA6⟩3.0.CO;2-1.
[39] C. Cooper, On the distribution of rank of a random matrix over a finite field, Random Struct. Algorithms 17 (3-4) (2000) 197–212, doi:10.1002/1098-2418(200010/12)17:3/4⟨197::AID-RSA2⟩3.0.CO;2-K.

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 Queen's University, Canada, respectively. He has also taught many courses and laboratories while working as a lecturer at KFUPM. His research interest include Wireless Sensor Networks data reliability; distributed storage systems, Network Coding and data dissemination, and information centric networking. He 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. He has received several recognitions and best papers 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 senior member of the IEEE, and is a former chair of the IEEE Communication Society Technical Committee on Ad hoc and Sensor Networks (TC AHSN). He is an IEEE Communications Society Distinguished Speaker (Distinguished Lecturer 2008–2010).