Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2012, Article ID 524543, 16 pages
doi:10.1155/2012/524543

Research Article

Information Fusion-Based Storage and Retrieve Algorithms for WSNs in Disaster Scenarios

Zhe Xiao,[1] Ming Huang,[2] Jihong Shi,[2] Wenwei Niu,[2] and Jingjing Yang[2]

[1] *School of Electrical and Electronic Engineering, Nanyang Technological University, Western Catchment Area, Singapore 639798*
[2] *School of Information Science and Engineering, Yunnan University, Kunming 650091, China*

Correspondence should be addressed to Zhe Xiao, zxiao1@e.ntu.edu.sg

Received 12 July 2011; Revised 13 September 2011; Accepted 19 September 2011

Academic Editor: Yuhang Yang

Sensor networks are especially useful in catastrophic or disaster scenarios such as abysmal sea, floods, fires, or earthquakes where human participation may be too dangerous. Storage technologies take a critical position for WSNs in such scenarios since the sensor nodes may themselves fail unpredictably, resulting in the loss of valuable data. This paper focuses on fountain code-based data storage and recovery solutions for WSNs in disaster scenarios. A review on current technologies is given on challenges posed by disaster environments. Two information fusion-based distributed storage (IFDS) algorithms are proposed in the "few global knowledge" and "zero-configuration" paradigm, respectively. Correspondingly, a high-efficient retrieve algorithm is designed for general storage algorithms using Robust Soliton distribution. We observe that the successful decoding probability can be provisioned by properly selecting parameters—the ratio of number of source node and total nodes, and the storage capacity M in each node.

1. Introduction

WSNs have attracted a lot of attention recently due to their broad applications. With the rapid development of microelectromechanical system, sensor nodes can be made much smaller with less cost. "Smart dust", as a form of WSNs, will become one of the most potential applications in real world [1]. They can be deployed in tragedy, isolated, and obscured fields to monitor objects, detect fires, temperature, flood, and other disaster incidents such as earthquakes, landslides, and ice damage. Sensing networks are ideal for such scenarios since conventional sensing methods that involve human participation within the sensing region are often too dangerous. These scenarios offer a challenging design environment because the nodes used to collect and transmit data can fail suddenly and unpredictably as they may melt, corrode, or get smashed. Hence, it is necessary to design reliable storage strategies to collect sensed data from sensors before they disappear from the network.

In 2006, Kamra et al. [2] designed and analyzed techniques to increase "persistence" of sensed data based on growth code—a variant of fountain code. Later on, Lin et al. [3] proposed an algorithm that uses random walks with traps to disseminate the source packets in the WSNs. They employed the Metropolis algorithm to specify transition probabilities of the random walks. However, the knowledge of the total number of sensors N, sources K, and the maximum node degree of the graph are required in their works. Recently, Aly et al. [4] proposed two new decentralized algorithms with limited or no knowledge of global information based on raptor codes. They afterward proposed two distributed flooding-based storage algorithms [5] for a WSNs wherein all nodes serve as sources as well as storage nodes, and the results demonstrated that it is required to query only 20–30% of the network nodes in order to retrieve the data collected by the N sensing nodes, in such a specific scenario where the buffer size is 10% of the network size. As a conclusion, a review on fountain code-based storage technologies is elaborated in Section 2.

The ultimate goal of any storage strategies is to get the maximum data recovering possibilities while encountering loss of data. A storage strategy needs to include two parts, that is, "how to store" and "how to retrieve". The two parts should be well matched with each other just like a

decoding algorithm needs to be suitable for the encoding process. However, most of the previous works only focus on the storage part involving how to network the backup of sensing packages to each storage node and how to process the back-up packages in distributed way. In this paper, we extend the works to a complete solution including both storage and recovery sections. With respect to the storage part, two IFDS algorithms are proposed in the "limited global knowledge" and "zero-configuration" paradigm, respectively. For the recovery part, a belief propagation and Gaussian elimination-based recovery Algorithm (BGRA) is designed for data retrieve in close connection with the proposed storage algorithms, and it is suitable for any storage algorithms using Robust Soliton distribution. Moreover, the general scenarios with consideration of the percentage of source node number K among total number of nodes N and the storage capacity M of each node are studied. We analyze in detail how the three parameters affect the data retrieve. The results indicate that a WSN with designable successful decoding probability can be deployed by selecting proper N, K, and M, which lays some foundation for the application of WSNs in disaster scenarios.

2. Fountain Code-Based Storage in WSNs

Fountain codes are a new class of rateless codes with finite dimension and infinite block length. The first class of efficient universal fountain codes was invented by Luby [6] and is called LT codes. The codes are designed for channels with erasures such as internet, but many distinctive characteristics make the codes become an excellent solution in a wide variety of situations. MacKay [7] mentioned two major applications in his review of fountain code, one is for broadcast and the other is for storage. In storage applications, fountain codes can be used to spray encoded packets as backup of a file on more than one storage device so as to prevent data loss caused by catastrophic failures of unreliable storage device; and to recover the file, one simply needs to gather enough packets from any intact devices and skip over the corrupted packets on the broken devices. Actually, the distributed storage model in WSNs is very similar to the case, and it seems easier to implement in WSN since the communication network used to bridge nodes makes it convenient to spray the back-up packages. In a sensor network, the storage device is the node with storage units. In order to prevent data loss caused by unexpected failure of the storage node, a similar solution is to network the important sensed data to multiple storage nodes, encoding them distributedly using fountain code and to store them as a backup. The original sensed data can be retrieved as long as to query enough storage nodes with enough encoded packages.

Based on these points, many researchers follow closely with fountain code-based decentralized storage technology and give specialized solutions to the storage problems of WSNs in disaster environments [2–6, 8]. The basic approach is to achieve distributed encoding in each storage node using simple exclusive-or operations. Specific implementations adopt the encoding process of growth code [2], LT code [3], or raptor code [4], respectively.

The way to disseminate the original sensed data to each storage node assumes crucial role in recovery performance of storage data in WSNs. Random walks [3, 4] and flooding [5] are two major ways to spray sensed data. The flooding dissemination adopts a very simple operation that each node floods the sensed data to all its neighbors and decides whether to store or discard the received packages according to the probability computed by random algorithms. Random walks employ Metropolis algorithm to disseminate the source packets. The number of random walks launched from each sensing node and the probabilistic forwarding tables for random walks are computed by the Metropolis algorithm. As long as a source block stops at a node at the end of the random walk, this node will store this source block. After all source blocks are disseminated, each storage node generates its encoded block. The basic features of the methods are summarized in Table 1.

According to global information requirements, the storage algorithm can be classified into two categories—"limited global knowledge" or "zero-configuration" algorithms [3–5]. If each node in the network knows the value of K—the number of sources, and the value of N—the number of storage nodes as a prerequisite for the designed storage algorithm, then the algorithm works in the "limited global knowledge" paradigm. However, in many scenarios, especially, when changes of network topologies may occur due to node joining-in or node failures, the exact value of N may not be available for all nodes. On the other hand, the number of sources K usually depends on the environment measurements or some events, and thus the exact value of K may not be known by each node either. As a result, to design a fully distributed storage algorithm which does not require any global information with "zero configuration" is very important and useful. In previous literatures, exact decentralized fountain codes (EDFC) and approximate decentralized fountain codes (ADFC) [3], distributed storage algorithms (DSA)-I [5] and raptor codes based distributed storage (RCDS)-I [4] are "limited global knowledge" based algorithms, and distributed storage algorithms (DSA)-II [5] and raptor codes based distributed storage (RCDS)-II [4] functions in "zero configuration". In the mode of "zero configuration", random walks can be used to estimate the network scale and further to approximately compute N, K in order to decide how to set the TTL segment (or maximum hop) of the package.

Viewed from the perspective of the recovery behavior, LT codes or Raptor codes based storage algorithm adopts the belief propagation (BP) process, which is recommended by Luby for decoding of fountain codes [6], as the recovery algorithm due to its low complexity. However, even though BP algorithm is simple and easy to implement, it does not explore all the encoding information in generator matrix G, so we do some amelioration on BP algorithm for a full exploitation of all the encoding information to enhance the retrieve performance. The detailed description is presented in Section 5. In addition, Growth code takes two situations into account—full recovery or optimal partial recovery.

Table 1: Comparison of flooding- and random walks-based data dissemination.

	Flooding	Random walks
Communication overhead	Big communication overhead	Small communication overhead
Global information requirements	Global Information is not required. Flooding disseminations can work in zero configuration combined with certain estimation of the global information	The global information of the total number of sensors N, sources K, and the maximum node degree of the graph are required. (However, the specific algorithm can be used for estimation of global information)
Implementation complexity	Simple and easy operations	Complex operations
Degree distribution guarantee	Distributed encoding process can be well guaranteed to satisfy Robust Soliton distribution	Using specific strategies to guarantee or approximate the Robust Soliton distribution

The goal is trying to completely recover all the storage data, but while it does not achieve full recovery, then pursuing the maximum of partial recovery to retrieve more storage data. This is a reasonable consideration in storage application. Inspired by these works, we proposed the information fusion based storage and retrieve algorithms for WSNs in disaster scenarios. Compared with the classic WSN paradigm, the major advantages using IFDS are the following. (i) IFDS algorithms adopt "flooding" to achieve data dissemination task, and each node never needs to keep a route table to sink node. Hence, each node never needs to rebuild new routing and update the route table due to failure of the nodes on the path to sink node. It is suitable for application in disaster environment. (ii) Over the various paths to sink node by flooding, each path transmits the "supplementary data" which includes not only the data information but also its relationship with other encoded blocks, so it has certain degree of redundancy, serving as a kind of "back up" for storage purpose. However, these advantages are at the cost of increasing a degree of communication overhead, computational overhead, and complexity.

With respect to the previous algorithms, several improvements are provided by IFDS algorithms. In the storage block assembling phase, the proposed IFDS algorithms do not need judge to accept or reject a data packet every time while the packet arrives a node, and each node in WSN calculates its own degree in preprocessing phase, which does once for all. In data dissemination phase, IFDS has a robust estimation method for hop segment $C_{\text{hop}}(s_{Si})$ in order to guarantee the desired dissemination task, so that all the sensed data can arrive at each node at least one time. Moreover, most of present algorithms never consider the node storage capacity which is actually an important factor affecting the data retrieve performance. The IFDS algorithms adopt a simple storage model for each node and give a detailed analysis on its behavior on data retrieve. Overall, the algorithms enhance the successful decoding probability, and we observe that a WSN with designable successful decoding probability can be deployed by selecting proper parameters—the ratio of source node number K and total number of nodes N, and the storage capacity M of each node. In what follows, the algorithms are discussed in detail.

3. Preliminaries and Modeling

In a real WSN, sensors are usually classified into several types based on different functions they assume. In Figure 1, we classify the set of sensors into the three kinds. Source sensors are located in the area where we expect to monitor for specific application. Source sensors are able to perform monitoring and generate packets of sensed data. Relay sensors collect received data into their buffer memory and are able to produce encoded blocks based on the distributed storage algorithm. Collector sensors or base station represents one or small number of WSNs nodes that are connected with the external network. The collector nodes is to collect data from their neighbors, recover the set of K source packets $\{B_{s1}, B_{s2}, \ldots, B_{sk}\}$ that originated at the source sensor nodes during a single time period, and forward this data to a database in the external network. Three classes of sensors are equipped with memory for data storage. Suppose that the WSN consists of N nodes that are uniformly distributed at random in a three-dimensional region. Among these N nodes, there are K source nodes that have information to be disseminated throughout the network for storage. The K nodes are uniformly and independently chosen at random among the N nodes. If all the nodes are assigned with sensing tasks, then K is equal to N, so the model is a general model which covers all the application scenarios with different ratio of N and K. We assume that no node has knowledge about the locations of other nodes and no routing table is maintained. Moreover, except the information of neighbor nodes, we assume that each node has limited or no knowledge of global information, working at "limited global knowledge" or "zero-configuration" status. The limited global information refers to the total number of nodes N and the total number of source nodes K. Any further global information, for example, the maximal number of neighbors in the network, is not available. In order to illustrate the two algorithms clearly, we will use the definitions of Node Degree $d_n(u)$ and Code Degree $d_c(y)$ in [5]. In order to achieve a better recovery performance, we hope that source data can be stored with balance in each node as a backup. Therefore, it is required to make data coming from different source nodes fused and stored throughout the network. These data stored

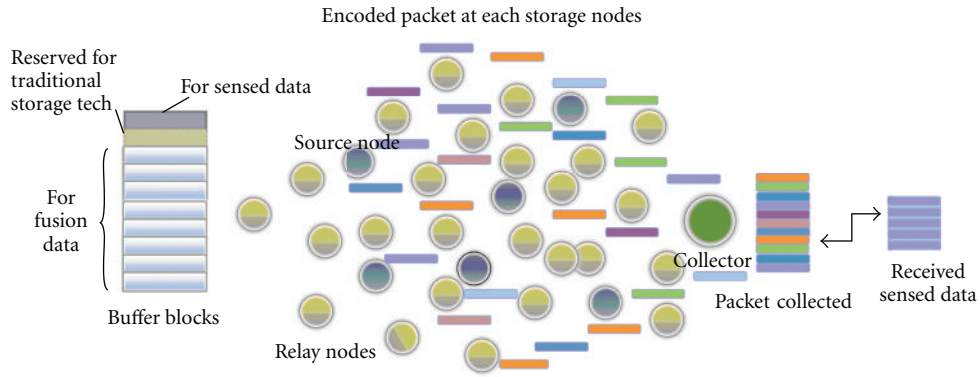


Figure 1: Illustration of information fusion-based distributed storage for WSNs.

in each node should be complementary and correlated. The process of data fusion is similar to the fountain code based on exclusive-or operations, but the process of encoding is a decentralized treatment in each node, so it works in a distributed way. Robust Soliton distribution [6] is adopted in our algorithms. In order to analyze the effect of the node capacity on the successful decoding probability as well as to make implementation of our algorithm easier, we model the node buffer as follows. The total buffer at each node is divided into several blocks according to the principle that each block has the size of sensed data package as a unit. For the source node, the first storage block is used for storing its own sensing data, the rest for distributed storage. For the relay nodes, all the blocks are used to store fused data. Each node reserves one storage block for traditional storage technology; in our implementation, we just simply store a selected packet copy with degree one. The buffer model is shown in Figure 1. Each block is labeled with one code degree to store fused data of certain several source packets. It is convenient to achieve the buffer model in practical applications using the existing hardware and software storage technology. In disaster areas, two cases of the node failure might happen. One is that a few nodes disappear due to out-charge, smash, or other reasons; the other is mass destruction or failure of nodes in a local area. Our storage strategy is designed for both scenarios.

4. Data Dissemination and Decentralized Storage

The concept of fountain code is a basis for the decentralized storage in WSNs. Although it is centralized in many coding literatures, we want to give a brief introduction of LT codes, especially concerning the Robust Soliton distribution for a better discussion on our storage and recovery algorithms in the following sections. Two storage algorithms are then presented to explain how to disseminate the sensed data and how to implement the encoding of LT codes at each storage node in decentralized way.

4.1. A Brief Introduction of LT Codes and Degree Distribution. Fountain is a class of erasure codes capable of reaching optimal erasure recovery on the binary erasure channels without fixing the rate. Fountain codes have remarkably simple encoding and decoding algorithms. In order to create an encoded symbol, an encoding host runs the encoding process as follows. Firstly, randomly choose the degree d_n of the packet from a degree distribution $r(d)$; the appropriate choice of r depends on the source file size k. Then, choose, uniformly at random, d_n distinct input packets, and then the sum of these input symbols over a suitable finite field (typically F_2) comprises the value of the encoded symbol. The concrete process of generating an encoding symbol using LT codes consists of three simple steps as follows [6]:

(i) randomly choose the degree d of the encoding symbol from a degree distribution. The design and analysis of a good degree distribution is a primary focus of the remainder of this paper;

(ii) choose uniformly at random d distinct input symbols as neighbors of the encoding symbol;

(iii) the value of the encoding symbol is the exclusive-or of the d neighbors.

Each encoding symbol has a degree chosen independently from a degree distribution. Degree distribution $\rho(d)$ is the probability that an encoding symbol has degree d. Luby gives two degree distributions—the Ideal Soliton distribution and its melioration—the Robust Soliton distribution.

Definition 1 (Robust Soliton distribution [1]). For constants $c > 0$ and $\delta \in [0, 1]$, the Robust Soliton distribution $\mu(i)$ is given by

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta}, \quad \text{for } 1 \le i \le k, \tag{1}$$

where $\beta = \sum_{i=1}^{k} (\rho(i) + \tau(i))$.

Here, $\rho(i)$ is Ideal Soliton distribution which is a probability distribution over $1 \le i \le k$; $\rho(i)$ and $\tau(i)$ are given by

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

$$\tau(i) = \begin{cases} \frac{S}{ik}, & \text{for } 1 \le i \le \frac{k}{S} - 1, \\ \frac{S \ln(S/\delta)}{k}, & \text{for } i = \frac{k}{S}, \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

The parameter S represents the average number of degree one code symbols and is defined as

$$S = c \cdot \sqrt{k} \cdot \ln\left(\frac{k}{\delta}\right). \quad (3)$$

As for decoding system, the most used is the BP algorithm and GE algorithm. BP algorithm first searches all degree-one symbol or say the column j that contains only one 1. And then all the 1's in its rows are canceled and their T_i are xored with T_j. The above process is iterated until the matrix G becomes all-0 matrix (decoding success) or until no more degree-one symbols can be found (decoding failure). BP is simple and fast, but while it encounters decoding failure, it may never use all the encoding columns of encoding G matrix as this causes a waste of part of encoding information. GE decoding algorithm adopts Gaussian elimination to solve problem $T = SG$ over typically F_2, which consists of two steps: triangularization step and back-substitution step. In the triangularization step, the goal is to convert the given matrix using row operations to upper triangular matrix. If the triangularization step is successful, then the back-substitution step can proceed by converting the triangular matrix into the identity matrix after which the GE is successfully finished. GE could exert a higher successful decoding probability with smaller overhead but costs more time to complete decoding process due to its high complexity. BP algorithm is recommended by Luby as decoding methods for LT codes [5], which depends on the specific encoding process and degree distribution function of LT codes. Our study finds that Robust Soliton and Ideal Soliton distribution have a common characteristic. The probability of degree 2 is the greatest; the sum probability of smaller degrees is usually greater, or say that most probability exists at small degrees. Such a degree distribution provides a large probability for directly obtaining a degree-one column in each iteration of BP process. Ideal Soliton and Robust Soliton distribution with $k = 100$ are shown in Figures 2 and 3, respectively. We can see degree 2 is greatest with possibility around half maximum. Comparative larger possibility is distributed at smaller degrees such as degree 1, 2, 3, and 4. Based on these observations, BP algorithm can be basically adopted as the recovery algorithm for WSNs; however, in order to obtain a high successful retrieve probability and while never fully recovering all the data and then trying the best to retrieve more data, we make some improvements over the BP algorithm. The part of works are focused in Section 5 concerning the recovery algorithm.

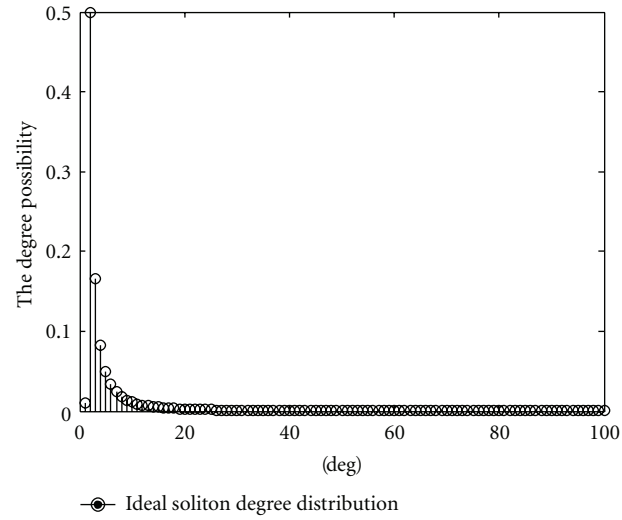


Figure 2: The Ideal Soliton degree distribution with $k = 100$.

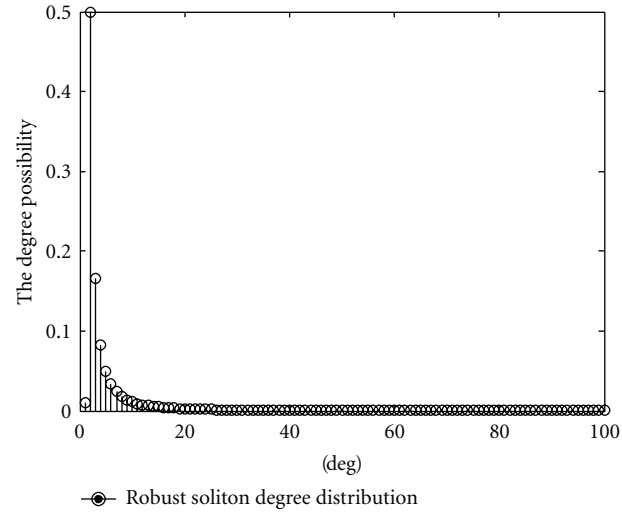


Figure 3: The Robust Soliton distribution μ (0.01, 0.01) with $k = 100$.

4.1.1. IFDS-I. In IFDS-I, each node in the network knows the limited global information N and K. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of sensing nodes or source nodes that are distributed randomly and uniformly in a field. Each of both source node and relay node acts as a storage node. Every node does not maintain routing or geographic tables, and the network topology G is not known. Every node can send a flooding message to the neighboring nodes and can detect the total number of neighbors by broadcasting a simple keep-alive message. Here we adopt flooding as the data dissemination method in order to satisfy the Robust Soliton distribution with reliability, based on which the proposed recovery algorithm can develop its advantages. We define the

IFDS-I Storage Algorithm

WHILE For each node in WSN $S = \{s_1, s_2, \ldots s_n\}$
  DO {Receive packets from neighbors $N(s_i)$}
    {**IF**$_1$ (*hop segment* $\neq 0$)
      {**IF**$_2$ (The node code of the received packet *belongs to* elements in $d_{c(si)}$.)
        {**IF**$_3$ (It is the first time to receive the packet B_{si})
          {Then accepting the packet into corresponding storage unit according to $d_{(n)}$ & $d_{c(si)}$. If there is no other packets stored in storage block before, then store it directly.}
        ELSE
          {M_j = exclusive-or (M_j, B_{si}), j depends on $d_{(n)}$ & $d_{c(si)}$.}
        ENDIF$_3$}
      ELSE
          {Put the packet into forward queue, set *hop segment* as $C_{\text{hop}}(B_{si}) = C_{\text{hop}}(B_{si})$-*1*.}
      ENDIF$_2$}
    ELSE
      {Discard the packet B_{si}, and no node to send to. }
    ENDIF$_1$}
ENDWHILE

Algorithm 1: The algorithm flow of IFDS-I.

average degree of the topology G as $d_{\text{mean}}(G)$. Each of the nodes in WSN except the collector nodes calculates its own code degree for its buffer blocks based on Robust Soliton distribution, which does once for all. The algorithm flow is illustrated as follows.

(i) *Input*. A sensor network $S = \{s_1, s_2, \ldots, s_n\}$ with N nodes; There are K source nodes which can produce source sensed packets $\{B_{s1}, B_{s2}, \ldots, B_{sk}\}$.

(ii) *Output*. Storage buffer blocks $\{M_1, M_2, \ldots, M_m\}$ for all sensors in S. Fused data are stored at each node according to node buffer model.

(iii) *Preprocessing*. *Step i.* Choose randomly the code degree of the encoding packets from degree distribution function, structure a set $d_{(n)}$ with m elements for m buffer blocks at each node, $d_{(n)} = \{d_{M1(n)}, d_{M2(n)}, \ldots, d_{Mm(n)}\}$. *Step ii.* Choose uniformly at random distinct source packets as degree distribution neighbors (ddn), $d_{c(si)}$ for each buffer blocks. $d_{c(si)} = \{(\text{ddn for } M_1), \ldots, (\text{ddn for } M_m)\}$. *Step iii.* Keep alive link with neighbors and generate a set of neighbors $N(s_i)$ using flooding and then obtain node degree $d_{n(si)}$. For each source node $S = \{s_1, \ldots, s_n\}$, generate source packet = node code, data style, hop, sensing data, flood to all of its neighbors $N(s_i)$, and set hop segment or TTL segment as $C_{\text{hop}}(B_{si}) = \lceil N/d_{\text{mean}}(G) \rceil - 1$. $d_{\text{mean}}(G)$ can be approximated by the node degree $d_{n(si)}$ of any arbitrary node s_i while the network is deployed with nodes of high density. $C_{\text{hop}}(B_{Si})$ is a very important parameter for a desired data dissemination. The sensed data from source nodes are desired to arrive at each node at least one time throughout the network; thus, the storage encoding process could be implemented rigidly according to the designed distribution degree function. Node code is a number that marks and distinguishes nodes in the WSN; Data style is used to tell the storage sensing data from other data; Hop is a data segment like TTL in TCP/IP protocol, which stands for the maximum hop of a packet. Compared with Aly's algorithms [5], the IFDS algorithms do not need to flip a coin to accept or reject a packet every time while a packet arrives a node, and each node in WSN calculates its own degree in preprocessing phase, which does once for all.

In order to show the data dissemination performance, we give an simple example for a WSN with parameters $N = 10$, $k = 5$. We define two matrix G_c and G_d. G_c is used to indicate the communication connectivity status between nodes in the WSN, it has N rows and N columns, and the matrix element $G_c(i, j)$ is a connectivity status with value 0 or 1; if $G_c(i, j) = 0$ $(i = 1, 2, \ldots, N;\ j = 1, 2, \ldots, N)$, it means no direct connection exists between the nodes N_i and N_j; on the contrary, if $G_c(i, j) = 1$, it means there exists a direct connection between the nodes N_i and N_j; in the diagonal, we use 0 as default. G_d is used to record how the process of data dissemination goes, the matrix element $G_d(i, j)$ $(i = 1, 2, \ldots, N;\ j = 1, 2, 3, \ldots, k)$ recodes the times of the sensed data from source node S_{Sj} arriving at node N_i.

In the example, we assume that $N_1, N_2, \ldots, N_5$ are source nodes which are rewritten as the standard form $\{S_1, S_2, \ldots, S_5\}$. At the beginning of flooding, all the source

BGRA Recovery Algorithm

WHILE ~ *terminus* (**Set** *terminus* = 0)
{**IF**$_1$ (There exists one degree column in G)
{Record sequence number of such columns into vector *rcdc*.}
ELSE
{**IF**$_2$ (BP algorithm found one degree column earlier)
{Construct $G_gS_r = T_r$;}
ELSE
{Construct a linear equations $GS = T$ and launch GE algorithm directly.}
ENDIF$_2$}
ENDIF$_1$}
{**FOR**$_1$ (all the elements $G(i, rcdc(j))$)
{**IF**$_3$ $(G(i, rcdc(j)) = 1)$
{Record sequences number i of rows into *rcdr*.}
ENDIF$_3$}
ENDFOR$_1$}
{**FOR**$_2$ (all the elements in *rcdr* and *rcdc*)
{$S(rcdr(i)) = T(rcdc(i))$;}
ENDFOR$_2$}
{**FOR**$_3$ (all the elements in *rcdc*)
{Set $G(i, rcdc(j)) = 0$ $(i = 1, \ldots, n)$;
{**FOR**$_4$ (all the elements in *rcdr*)
{**IF**$_4$ $(G(rcdr(i), j) = 1;)$
{$T(j) = x$ *or* $(S_r(rcdr(1, i)), T(j))$; }
ENDIF$_4$}
ENDFOR$_4$}
Set $G(rcdr(i), j) = 0; (i = 1, \ldots, m);$}
ENDFOR$_3$}
{**IF**$_5$ (The G becomes all-0 matrix;)
{Set *terminus* = 1;}
ENDIF$_5$}
{**IF**$_6$ (GE is used in the algorithm)
{Combine the decoding results from BP and GE.}
ENDIF$_6$}
ENDWHILE

Algorithm 2: The algorithm flow of BGRA.

nodes first calculate the hop segment value $C_{\text{hop}}(B_{Sj})$ $(j = 1, 2, \ldots, 5)$ for their flooding data blocks. In this case, all the source nodes obtain the same value, $C_{\text{hop}}(B_{Sj}) = (N/d_{n(si)}) - 1 = 1$. Then the source nodes fill the hop segment of data blocks with one and then flood the data to their neighbours. According to G_c, S_1 has neighbour nodes N_2, N_3, N_5, N_6, and N_{10}; S_1 floods the sensed data to these nodes, and G_d records the fact that the data block from S_1 arrives nodes N_2, N_3, N_5, N_6, and N_{10} one time. The working flow for other source nodes is the same. After the first flooding, G_d becomes as Figure 4(b). Next, every nodes which received the sensed data during the first flooding will further flood the data to their neighbours. Based on G_c, the obtained G_d is further updated as Figure 4(c). And at the point, the hop value of data block is decreased to 0, so the flooding terminates. We can see that there is no 0 element in the final G_d, which indicates that each sensed data block is guaranteed to arrive at each node at least one time. Therefore, the data dissemination task is perfectly completed, and the encoding process is also guaranteed for distributed storage.

In Figure 5, the red curve indicates the TTL of data packets or the number of transmissions required for a successful data dissemination, in order to assure that all the sensed data can arrive at each node at least one time. The blue curve is the actual TTL used by IFDS-I. It is obvious that the value on blue curve is equal or greater than the red one; this means that IFDS-I provides the data packets with a longer living life than enough, or say that the packets can arrive at more nodes. And we find that the counter of packet could actually be set as a number smaller than $N/d_{\text{mean}}(G)$, while WSN is a large network with high connectivity. The main operations of IFDS-I is shown in Algorithm 1.

4.1.2. IFDS-II. In IFDS-II, we assumed that N and K are known in advance for each node in the network. This might not be the case in practical disaster scenarios where the

G_c

		N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}
S_1	N_1	0	1	1	0	1	1	0	0	0	1
S_2	N_2	1	0	1	0	1	0	0	0	1	1
S_3	N_3	1	1	0	0	1	0	0	0	1	1
S_4	N_4	0	0	0	0	1	0	1	1	1	1
S_5	N_5	1	1	1	1	0	0	0	0	0	1
	N_6	1	0	0	0	0	0	0	1	0	1
	N_7	0	0	0	1	0	0	0	1	1	1
	N_8	0	0	0	1	0	1	1	0	1	1
	N_9	0	1	1	1	0	0	1	1	0	1
	N_{10}	1	1	1	1	1	1	1	1	1	0

(a)

G_d after 1st flooding

	S_1	S_2	S_3	S_4	S_5
N_1	0	1	1	0	1
N_2	1	0	1	0	1
N_3	1	1	0	0	1
N_4	0	0	0	0	1
N_5	1	1	1	1	0
N_6	1	0	0	0	0
N_7	0	0	0	1	0
N_8	0	0	0	1	0
N_9	0	1	1	1	0
N_{10}	1	1	1	1	1

(b)

G_d after 2nd flooding

	S_1	S_2	S_3	S_4	S_5
N_1	5	4	4	2	4
N_2	4	5	5	3	4
N_3	4	5	5	3	4
N_4	2	3	3	5	2
N_5	4	4	4	2	5
N_6	2	2	2	2	2
N_7	1	2	2	4	2
N_8	2	2	2	4	2
N_9	3	3	3	4	4
N_{10}	5	5	5	5	5

(c)

Figure 4: Data dissemination implemented in IFDS-I.

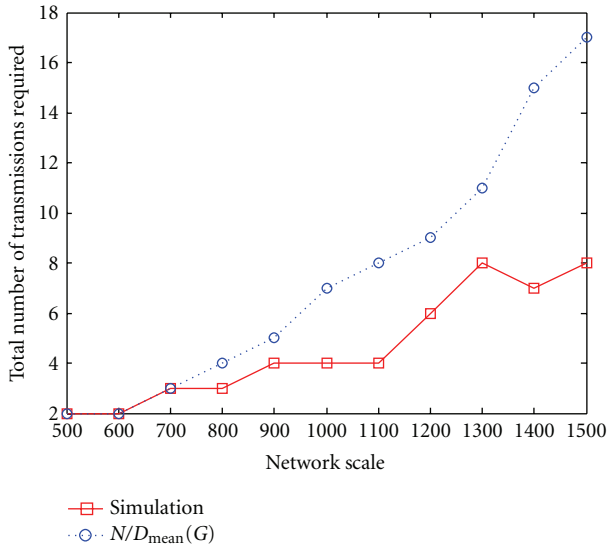


Figure 5: The relationship between network scale and C_{hop}.

change of connectivity status may occur between nodes due to the event of sensor failure or new nodes joining in. Therefore, we extend IFDS-I to IFDS-II that is totally distributed without knowing global information with "zero configuration". The idea is that each source node s_{Si} will estimate a value for its hop counter $C_{\text{hop}}(s_{Si})$ without knowing N and K. In IFDS-II, each source node s_{Si} will perform a hop-estimation phase that will calculate the value of the counter $C_{\text{hop}}(s_{Si})$. The hop-estimation process starts before data dissemination; however, the estimation not only works at the network startup; instead, it is dynamic process working throughout. The hop estimation is implemented while source nodes receive a keep-alive hop-estimation packet from their neighbors. A keep-alive hop-estimation packet consists of {node code, data style, estimated hop}.

Hop estimation: let s_{Si} be a source node in a distributed network. Each node s_{Si}, $(i = 1, 2 \ldots K)$ will dynamically determine value of the counter $C_{\text{hop}}(s_{Si})$. The node s_{Si} knows its neighbors $N(s_{Si})$ by keep-alive message. Each source node will independently decide a value for its counter by following several steps. *Step i.* Flood a keep-alive hop-estimation packet = {node code, data style, estimated hop} to its neighbors $N(s_{Si})$. Node code segment fills its own node code; data style is labeled by hop-estimation packet, and estimated hop is recorded with 0. *Step ii.* Each node forwards the hop-estimation packet to its neighbors with estimated hop value plus one while recieving mutiple the hop-estimation packets from the same source, the node compares them and chooce the smallest to plus one. *Step iii.* When a source node received its corresponding hop-estimation packets again, it compares the values of estimated hop segments of all the packets received and choose the maximum as N_{dia}. *Step iv.* Set $C_{\text{hop}}(s_{Si}) = \lceil N_{\text{dia}}/2 \rceil$. Thus, one iteration of hop estimation is finished. The estimation algorithm flow is illustrated in Figure 6.

In the flow chart, each source node should maintain a hop value. The hop value is used for a real-time estimation of the network size based on the already received hop-estimation packets. It will be updated when a packet with a bigger estimated hop segment value is received. Every time when the packet passes through a node, the value of the estimated hop segment will increase one; so, a bigger hop segment means that the packet can reach to a node in a father location and it has come back to the source node again. Thus, the packets with the biggest estimated hop segment are those reaching the edge of the network. Continuous updating of the hop value until achieving convergence can be used to approximate the network radius or network scale. In this process, if structure of WSN alters, such as nodes failures, movements, or network expanding, all these changes will be reflected in the estimated hop segment of hop-estimation packets, and further the hop value will be also updated adaptively. For example, if the network is expanding, the hop value will increase and become stable after several updates. Moreover, while new sensing nodes are assigned in WSN, the estimation process will be triggered for these new source nodes which start estimating and knowing the network size. Once the hop counts $C_{\text{hop}}(s_{Si})$ is approximated at each source node, the encoding operations of IFDS-II are similar to

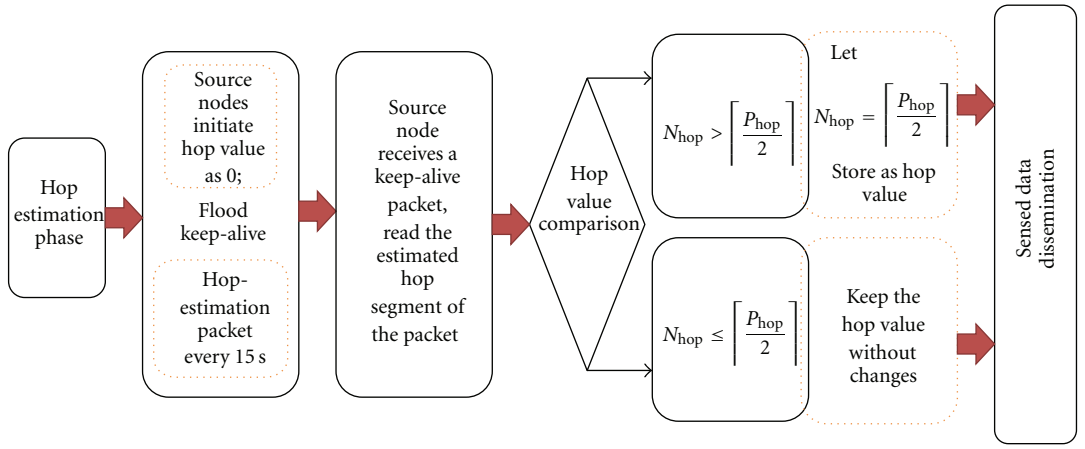


Figure 6: The hop-estimation process flow of IFDS-II.

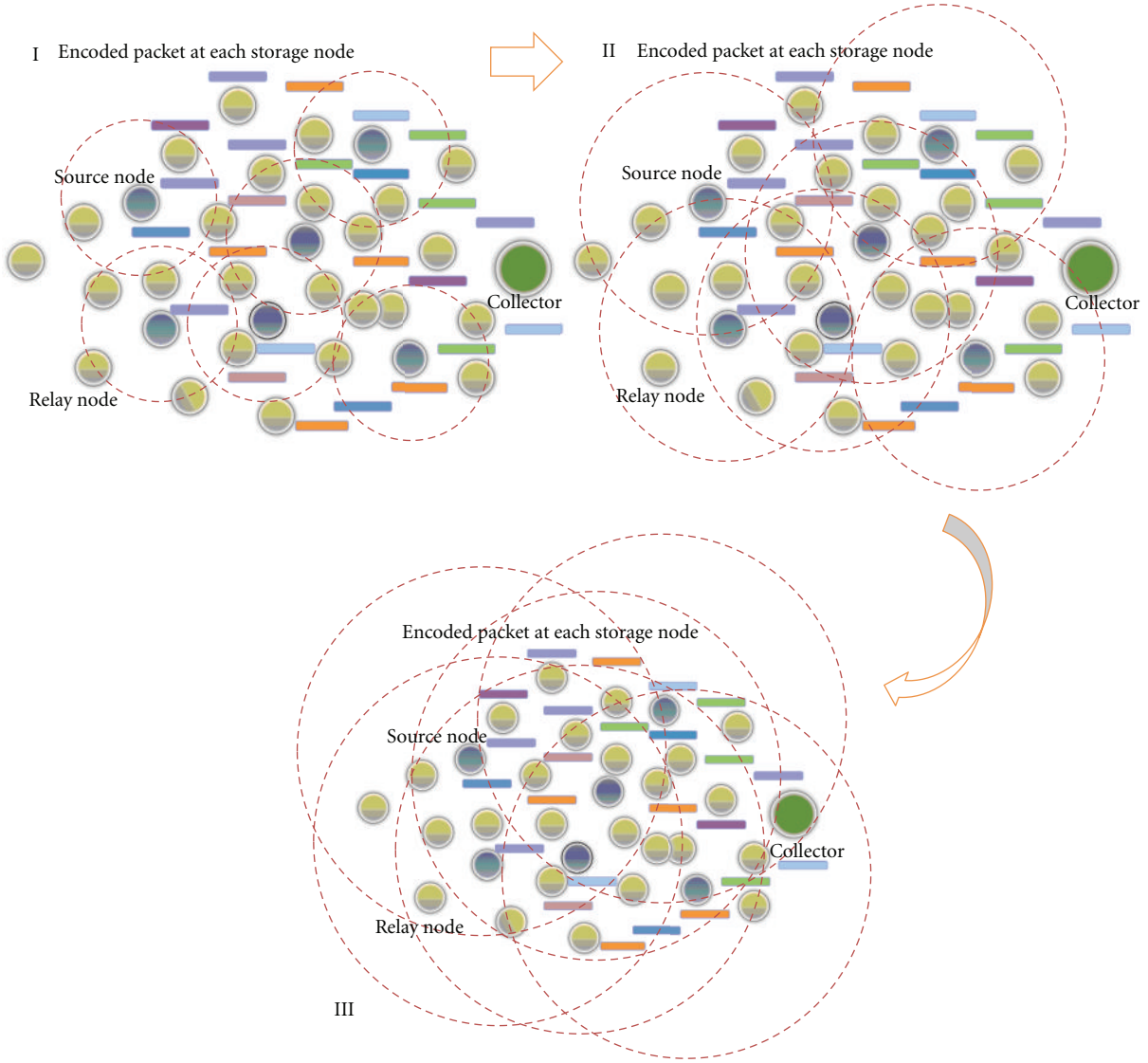


Figure 7: The data dissemination implemented by IFDS-II.

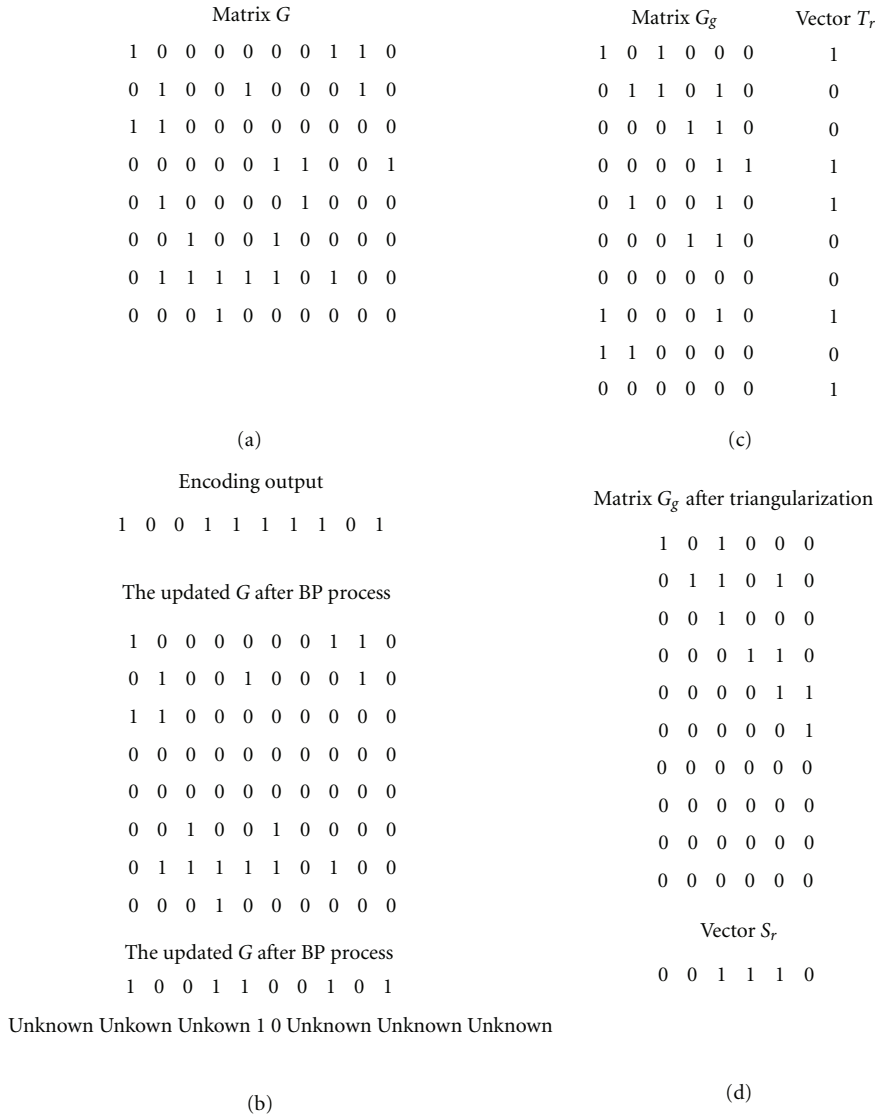


Figure 8: The original G matrix and the evolvement of G, T, and ST vector in the process of BGRA.

encoding operations of IFDS-I. In data dissemination phase, source nodes will assemble and flood the sensed data packets using the real-time hop value as the hop segment, and the completion evolution of dissemination task is shown as follows (Figure 7).

In the beginning stages (I, II), the maintained hop value is still never fully updated since the packets that spread to the network edge never come back; therefore, the assembled data packets cannot reach far from the source node due to a small TTL. Along with the dynamic estimation of the network size, the hop value is updated and continuously increasing; then the assembled data packet can reach farther in the process until the hop value reaches stable, and, at the point, the network radius or network scale can be correctly estimated; the packets flooded from source node can arrive at any nodes in the network. In our buffer model, one block is reserved for traditional storage method. Here we just randomly store one sensed packet with degree one using uniform distribution. The storage performance of both IFDS-I and IFDS-II will be elaborated in Section 6 combining with the recovery algorithm proposed in Section 5.

4.2. Power Consumption of IFDS Algorithms. The overall energy in the WSN nodes is consumed in three distinct processes: data processing, data transmission, and sensing tasks. The proposed IFDS algorithms will majorly affect the first two processes; certain overhead will be brought during implementation of the algorithms. The communication overhead is the major overhead due to wide flooding for data dissemination. However, on the other hand, since the algorithms never need to maintain the routing tables, so they would never introduce the routing overhead. The distributed storage encoding will cause certain computation power consumption. In order to decrease the computation complexity, the proposed IFDS algorithms do not need to flip a coin to accept or reject a data packet every time

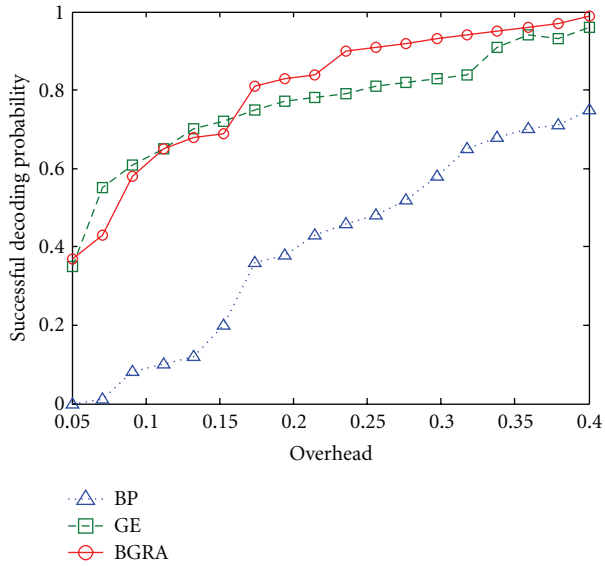


Figure 9: The successful decoding probability as a function of encoding overhead while $k = 100$ for LT code (0.01, 0.01).

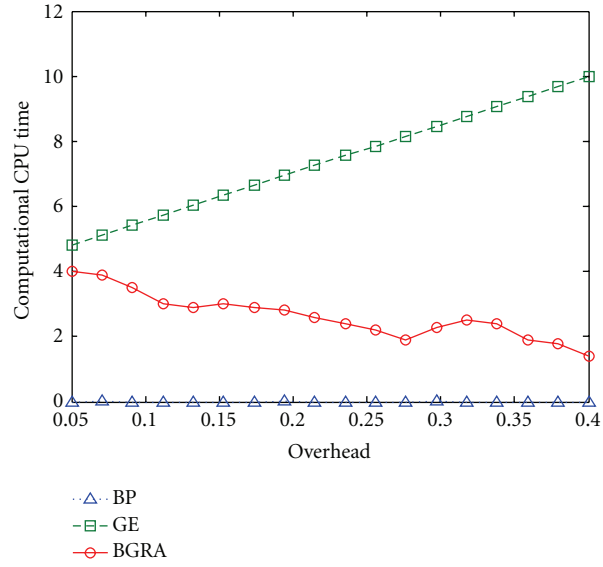


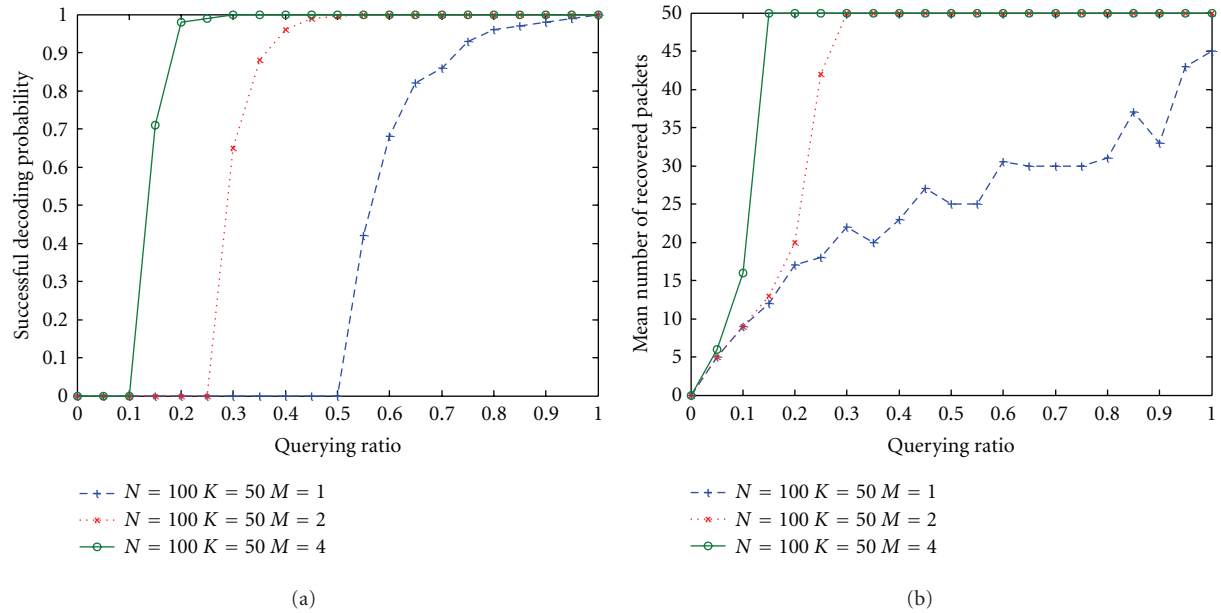
Figure 10: The computational CPU time as a function of encoding overhead while $k = 100$ for LT code (0.01, 0.01).

while the packet arrives a node; instead, each node in WSN calculates its own degree in preprocessing phase, which does once for all. This is more energy efficient than the previous algorithms that will calculate the code degree and make selection from every arriving packet for encoding. The typical energy management techniques with the basic idea, to shut down sensors when not needed and wake them up when necessary, can be easily applied and combined with IFDS algorithms, because IFDS can estimate the network size adaptively. Moreover, Since the deployment of the WSNs in difficult-to-access areas makes it difficult to replace the batteries of sensor nodes. The use of solar cells, super capacitors, or rechargeable batteries is necessary for the long-term sensor node operation. A long-term operation could be achieved by adopting a combination of hardware and software techniques along with energy efficient WSN design.

5. Recovery Algorithm

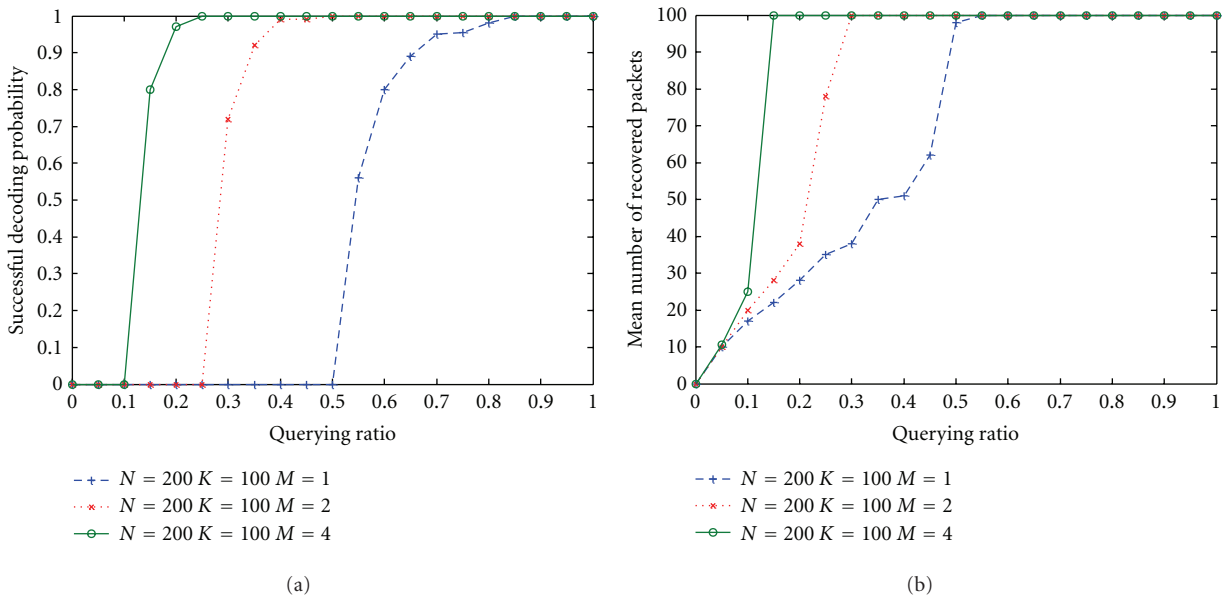
In Section 3, we refer that BP algorithm is generally used as a basic recovery algorithm to retrieve storage data in WSNs. However, the BP algorithm has some limitations while used for storage applications. BP algorithm is fast with low complexity but it must find degree-one column in generator matrix G for each iteration to make decoding process go ahead, which prohibits the improvement of recovery efficiency, because it may not use all the encoding relationship recorded in G while it cannot find degree-one column in process, or say that the BP algorithm never takes use of all the encoding information while encountering stop set which will greatly reduce the successful probability of complete recovery. Gaussian elimination (GE) is another typical way for decoding of LT codes, and now many improved algorithms are proposed for decoding of fountain codes [9–13]. In order to solve the problems caused by the stop set of the BP algorithm, we expect to combine BP and GE algorithms and to provide a good tradeoff between the advantages of both algorithms for a better recovery performance. The core idea is to first use BP algorithm with very low complexity directly to find degree-one column in G and then to consider adoption of GE operation to find the potential degree-one column from the remaining columns. It can overcome the negative influence of algorithm termination of BP and improve the decoding efficiency for LT codes through fully digging out encoding information from matrix G. We name the algorithm, Belief propagation and Gaussian elimination based Recovery Algorithm (BGRA). The algorithm flow is shown in Algorithm 2.

We give an intuition of how the BGRA works by considering a simple example. In the example, we set $N = 10$, $K = 8$, that is, there are 10 storage nodes in the WSN amongst which there are 8 sensing nodes assigned with the monitor tasks. We use one bit with 0 or 1 to stand for one sensed source packet produced at sensing nodes. In practice, a package is to carry a certain bit of information, but here this does not affect our description of this algorithm. The sensed data from the 8 sensing nodes is $S_o = \{0, 0, 1, 1, 0, 1, 1, 0\}$. After the distributed encoding process using Robust Soliton distribution μ $(c = 0.01, \delta = 0.01)$ is $\{1, 0, 0, 1, 1, 1, 1, 1, 0, 1\}$ which is stored in the storage unit of the 10 storage nodes, respectively, the matrix G that records encoding information such as code degree and neighbors is shown in Figure 8(a). BP algorithm is launched to decode firstly. It searches all the columns with one degree in G. For matrix G in Figure 8(a), only the tenth column is degree-one column. BP algorithm starts decoding from the column and set $s_4 = t_{10} = 1$, then the algorithm refreshes all the corresponding columns in G and updates the decoding input vector T as well as decoding output vector S_T according to the recovery algorithm. The updated G and T are shown in Figure 8(b). Since the



(a) (b)

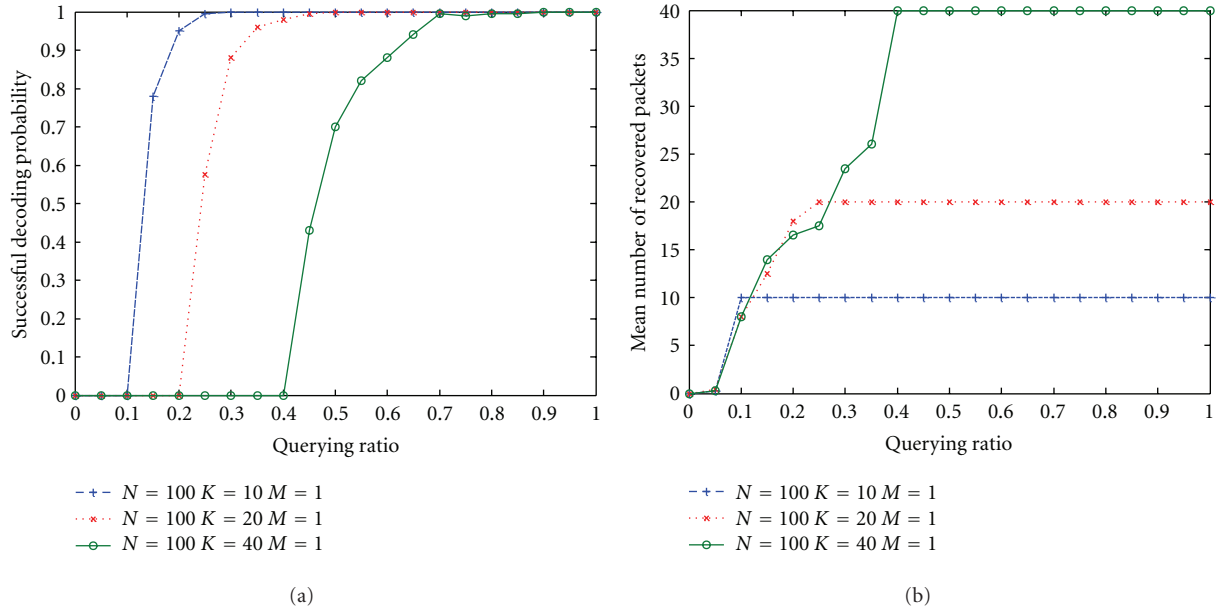
Figure 11: (a) The successful decoding performance; (b) the mean number of recovered packets as a function of querying ratio while $N = 100$, $K = 50$, $K/N = 50\%$, and M changes.



(a) (b)

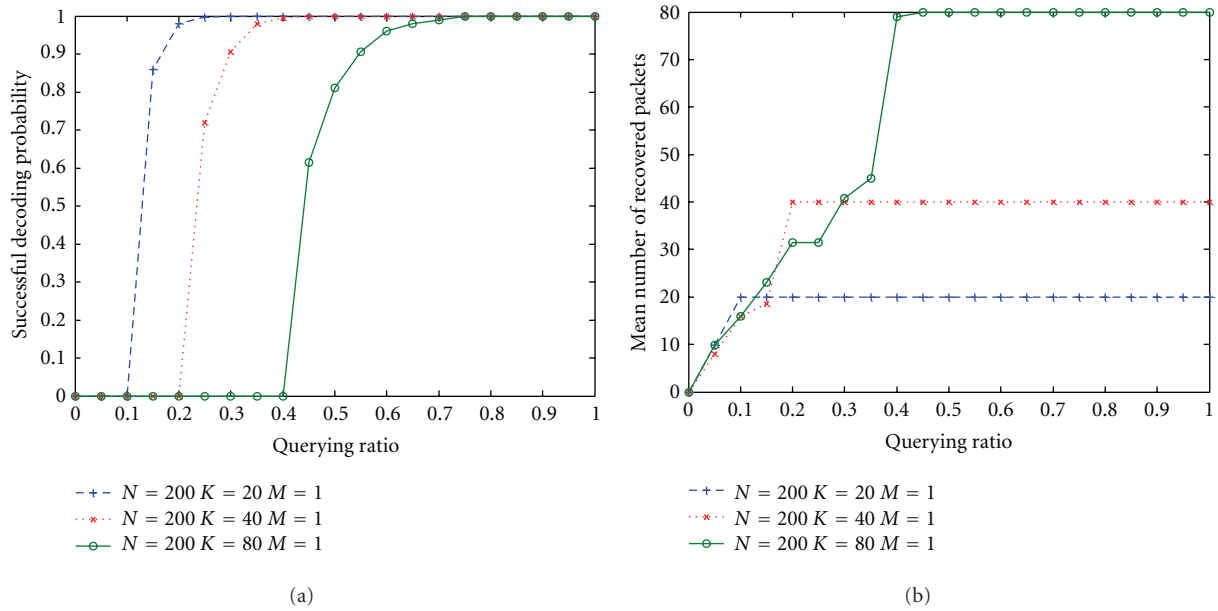
Figure 12: (a) The successful decoding performance; (b) the mean number of recovered packets as a function of querying ratio while $N = 200$, $K = 100$, $K/N = 50\%$, and M changes.

algorithm cannot find another degree-one column, BGRA launches GE algorithm to continue decoding based on the rest columns of G, which is the reason why BGRA can fully exploit encoding information from G. Before GE algorithm is started, linear equations $G_g S_r = T_r$ are constructed over the rest columns of G after BP process. G_g is a matrix that records code degree and neighbors of source symbols except those decoded during previous BP process. In order to construct G_g, the rows and columns with all-0 elements are deleted from G and its transpose is G_g. T_r is the updated T after deleting elements corresponding with those all-0 columns. In our example, G_g is the transpose of G after deleting the fourth row and the tenth column. T_r is T without the tenth element. G_g and T_r are shown in Figure 8(c). The deleted row and column has only 0 elements after the first iteration of BP algorithm. The matrix G_g after triangularization is shown in Figure 8(d) as well as the solution S_r of linear equations $G_g S_r = T_r$. The final step of BGRA algorithm is to combine both parts of recovery results from BP and GE and to recover the source symbols. The solution is very



(a) (b)

FIGURE 13: (a) The successful decoding performance; (b) the mean number of recovered packets as a function of querying ratio for networks with $N = 100$, $M = 1$ while the value of K/N changes.



(a) (b)

FIGURE 14: (a) The successful decoding performance; (b) the mean number of recovered packets as a function of querying ratio for networks with $N = 200$, $M = 1$ while the value of K/N changes.

simple; it just needs to orderly put all the elements of vector S_r into the places labeled "Unknown" in vector S_T. BGRA ends at this point, and the retrieve data is obtained as $S_T = \{0, 0, 1, 1, 0, 1, 1, 0\}$. In addition, the On the Fly GE algorithm proposed in [10] can be used for solution of $G_g S_r = T_r$ with a lower complexity.

BGRA algorithm, as integration of BP and GE algorithm, provides a good tradeoff between the advantages of both the BP algorithm and the GE algorithm. The BGRA algorithm has reasonable decoding complexity that is in between the low complexity of the BP algorithm and the high decoding complexity of the GE algorithm. The BGRA algorithm has a successful decoding probability that is comparable to that of the GE algorithm and significantly better than that of the BP algorithm. Additionally, the decoding CPU computational time of BGRA algorithm does not rapidly increase with overhead, as is the case for the GE algorithm.

In order to compare the three algorithms, we experiment BP, GE, and BGRA algorithms with different values of overhead for the number of sensed source data $k = 100$,

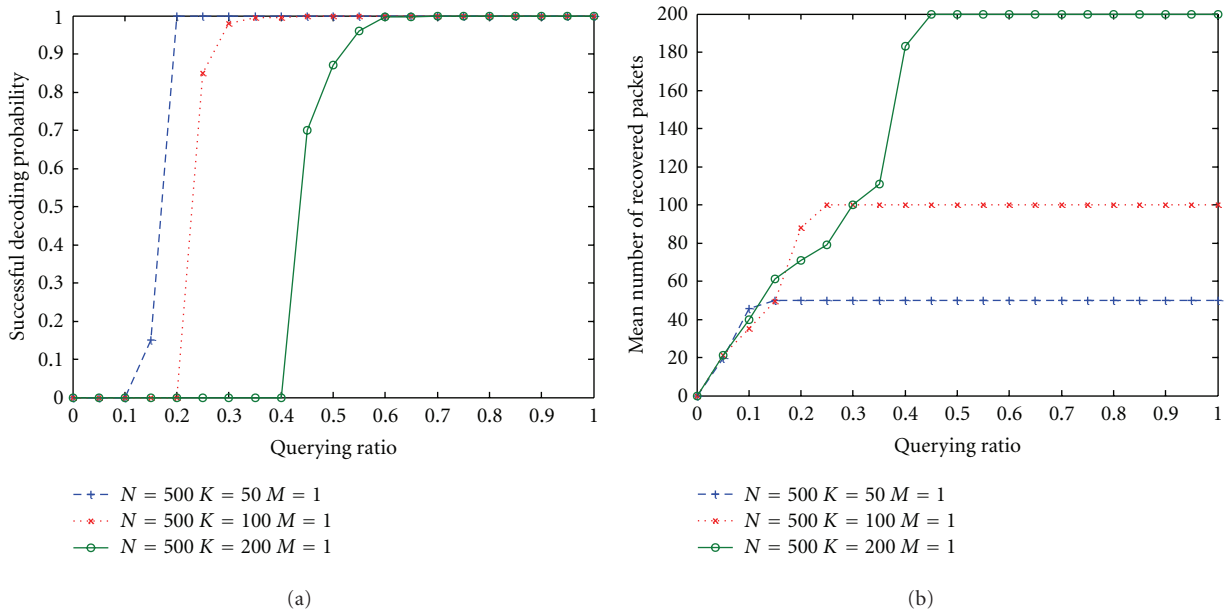


FIGURE 15: (a) The successful decoding performance; (b) the mean number of recovered packets as a function of querying ratio for networks with $N = 500$, $M = 1$ while the value of K/N changes.

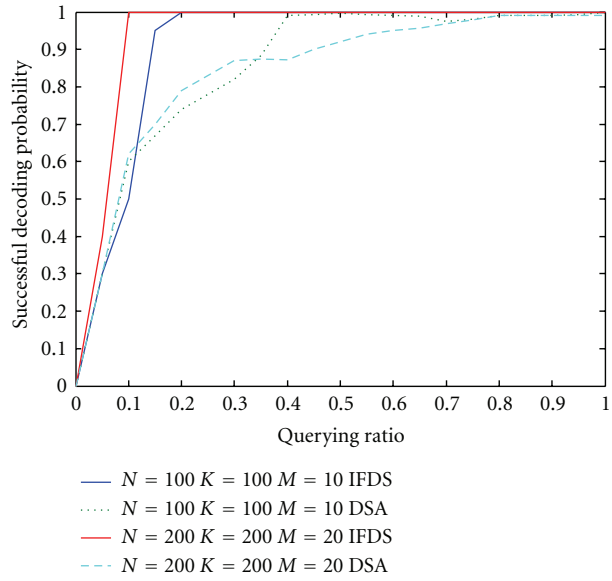
using Robust soliton degree distribution ($c = 0.01$, $\delta = 0.01$) for storage encoding. Assuming that the number of retrieved encoded storage packets is n, the overhead is defined as $q = (n - k)/k$. A bigger overhead means that more storage nodes are queried for data recovery. Figure 9 shows that BP algorithm performs poor while overhead is small, but GE and BGRA exert a higher decoding performance; while overhead is more than about 0.2, both will produce a successful decoding probability over 90%. In Figure 10, we show the computational CPU time Rt using BP, GE, and BGRA while changing overhead. Compared with Rt of BP which keeps stable along with overhead's increase, Rt of GE grows quickly while overhead increases. Rt of BGRA ranges between BP and GE with the trend that more overhead costs less time to decode. Based on the analysis above, BP algorithm actually functions poor on data recovery behavior. Taking the computational capacity of node and successful retrieve probability into account, BGRA can provide a good data recovery performance.

6. Results and Discussion

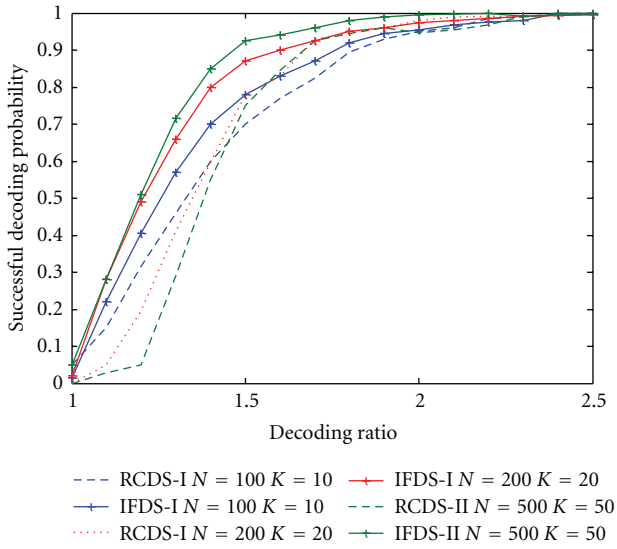
Figures 11 and 12 show the decoding performance with *IFDS-I* for different network scale with fixed ratio of $K/N = 50\%$ while M changes. Figures 13–15 illustrate the decoding performance with IFDS-II for different network scale while the value of K/N changes. Querying ratio is the percentage of the number of queried nodes q among the total number of nodes N. Successful decoding probability P is the probability that the K source packets are all recovered from the q queried nodes. The successful decoding probability for different scale networks with fixed $K/N = 50\%$ while M is 1, 2, 4 is shown in Figures 11(a) and 12(a). It is obvious that a bigger node buffer size can provide a better decoding performance; especially for a WSN while each node just has a small storage memory, to increase the storage capacity is a good way to improve the successful retrieve probability. Figures 11(b) and 12(b) illustrate the number of recovered packets at different querying ratio. Although a full recovery at the low querying ratio is never achieved, it could well give partial recovery of which performance is better than only using BP algorithm. The number of recovered packets is generally more than the number of queried nodes with a rapid increase along with the increase of querying ratio. The results show that the retrieve performance has a similar relationship with the querying ratio for any different scale networks while K/N is fixed.

Figures 13–15 illustrate the decoding performance with IFDS-II for different scale networks while each node has fixed one unit buffer block and K/N changes; a higher percentage of source node requires more queried nodes to retrieve the sensed data, which is in line with common sense. The results also show that 10% increase in the percentage of source node needs to query 10–20% more nodes in order to keep the same successful decoding probability. From Figures 13–15(a), we can see if the number of any queried nodes is slightly more than k; then it could achieve a full data retrieve. Based on these findings, we can easily deploy a WSN for monitoring in practical catastrophic environment.

Figure 16(a) compares the performance of IFDS-I with DSA-I [5]. DSA-I algorithm utilizes flooding and the node degree of each node to disseminate the sensed data from sensors throughout the network, and the encoded data are stored distributedly in each node for later data retrieve. Both IFDS and DSA algorithms consider the storage capacity of each node which is modeled with the number of buffer units. In [5], the authors analyze the data retrieve performance



(a)



(b)

Figure 16: (a) Comparison of decoding performance using IFDS-I and DSA-I; (b) comparison of decoding performance using IFDS and RCDS.

of DSA-I when the node buffer size is 10% of the network storage size. Under the same condition, we give our results and we find that IFDS has a better decoding performance. Compared with 20–30% querying nodes using DSA-I, IFDS-I only requires to query 15% nodes in network with $N = K = 100$ and 10% nodes in network with $N = K = 200$ for nice recovery of all sensed packages from source nodes. IFDS has a similar data-disseminating method and the same degree distribution function with DSA; however, a better recovery algorithm is used by IFDS-I, so IFDS-I functions better than DSA-I. Figure 16(b) shows the performance comparison between the IFDS and RCDS algorithms. RCDS-I are "limited global knowledge" based algorithms, RCDS-II works in "zero-configuration". In order to compare IFDS and RCDS, we use the definition of the decoding ratio in [4]. The decoding ratio is defined as the ratio between the number of querying nodes and the number of sources. It can be seen that IFDS algorithms can exert a higher successful recovery probability while querying the same number of sensor nodes for different network scale. While the decoding ratio is between 1.2 to 2, the successful recovery probability of IFDS algorithm is about 10–15% greater than RCDS algorithm. In addition, we can also observe that the IFDS algorithms perform better for larger scale WSNs.

7. Conclusion

This paper proposes two IFDS algorithms in the "few global knowledge" and "zero-configuration" paradigm, respectively. An efficient retrieve algorithm is designed correspondingly, which is generally suitable for the storage algorithms using Robust Soliton distribution. The algorithms enhance the successful decoding probability. The detailed results of data retrieve performance and its relationship with three parameters—the total number of nodes N, the number of sensing nodes K, and the number of storage units equipped at each node M, is studied, which shows that we can control the successful decoding probability through setting up desired network parameters.

Acknowledgments

The authors thank the Research Project from Communication Branch of Yunnan Power Grid Corporation, Training Program of Yunnan Province for Middle-aged and Young Leaders of Disciplines in Science and Technology (Grant no. 2008PY031), and the National Natural Science Foundation of China (Grant no. 60861002) for financial support.

References

[1] D. Butler, "2020 Computing: everything, everywhere," *Nature*, vol. 440, no. 7083, pp. 402–405, 2006.

[2] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: maximizing sensor network data persistence," in *Proceedings of the ACM SIGCOMM*, Pisa, Italy, September 2006.

[3] Y. Lin, B. Liang, and B. Li, "Data persistence in large-scale sensor networks with decentralized fountain codes," in *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM '07)*, pp. 1658–1666, Anchorage, Alaska, USA, May 2007.

[4] S. A. Aly, Z. Kong, and E. Soljanin, "Raptor codes based distributed storage algorithms for wireless sensor networks," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '08)*, pp. 2051–2055, Toronto, Canada, July 2008.

[5] S. A. Aly, M. Youssef, H. S. Darwish, and M. Zidan, "Distributed flooding-based storage algorithms for large-scale wireless sensor networks," in *Proceedings of the IEEE International Conference on Communications (ICC '09)*, Dresden, Germany, June 2009.

[6] M. Luby, "LT codes," in *Proceedings of the 34rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '02)*, Vancouver, BC, Canada, November 2002.

[7] D. J. C. MacKay, "Fountain codes," *IEE Proceedings Communications*, vol. 152, pp. 1062–1068, 2005.

[8] S. A. Aly, Z. Kong, and E. Soljanin, "Fountain codes based distributed storage algorithms for large-scale wireless sensor networks," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '08)*, pp. 171–182, St. Louis, Mo, USA, April 2008.

[9] S. Kim, K. Ko, and S. Y. Chung, "Incremental Gaussian elimination decoding of raptor codes over BEC," *IEEE Communications Letters*, vol. 12, no. 4, pp. 307–309, 2008.

[10] V. Bioglio, M. Grangetto, R. Gaeta, and M. Sereno, "On the fly Gaussian Elimination for LT codes," *IEEE Communications Letters*, vol. 13, no. 12, Article ID 5353274, pp. 953–955, 2009.

[11] D. Burshtein and G. Miller, "An efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2837–2844, 2004.

[12] W. Niu, Z. Xiao, M. Huang, J. Yu, and J. Hu, "An algorithm with high decoding success probability based on LT codes," in *Proceedings of the 9th International Symposium on Antennas Propagation and EM Theory (ISAPE '10)*, pp. 1047–1050, Guangzhou, China, November 2010.

[13] H. Zhu, G. Li, and S. Feng, "BPL decoding algorithm of LT code," *Computer Science*, vol. 36, no. 10, pp. 77–81, 2009.