This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2010 proceedings

Distributed Storage Coding for Flexible and Efficient Data Dissemination and Retrieval in Wireless Sensor Networks

Ning Cao[†], Qian Wang[‡], Kui Ren[‡], and Wenjing Lou[†]
[†]Department of ECE, Worcester Polytechnic Institute, Email: {ncao, wjlou}@ece.wpi.edu
[‡]Department of ECE, Illinois Institute of Technology, Email: {qian, kren}@ece.iit.edu

***Abstract*—The technique of distributed storage coding has been widely used in wireless sensor networks for increasing the robustness of data storage and efficiency of data retrieval. Existing works mainly focus on scenarios in which each storage node stores a linear combination of a subset of K data packets generated by different source nodes. By solving the linear equations, a collector can recover all the K data packets with high probability. This paper explores the problem of flexible and efficient data dissemination and retrieval in wireless sensor networks. Our scheme exploits the broadcast nature of wireless transmission and improves the traditional random walk algorithm for efficient data dissemination. Furthermore, through the use of Fountain codes in data encoding, it enables a mobile collector to recover up-to-date data generated by any subset of source nodes. Specifically, by querying any $t(1+\epsilon)$ storage nodes at the t-th time slot, the target data can be retrieved without having to decode all the source packets. Simulation results validate our analysis and show that the proposed schemes are flexible and efficient.**

I. Introduction

A wireless sensor network usually consists of a large number of sensor nodes with constrained communication, computation, and storage capabilities, and can be easily deployed to various terrains of interest to sense the environment. In many wireless sensor network applications, multiple and distributed sensor nodes usually generate a large amount of data continuously over their lifetime and these data may be retrieved later by collectors for further analysis. Powerful sink nodes are usually introduced to gather data and provide data access service. However, such centralized storage can lead to the single point of failure and easily attracts attacks. Moreover, centralized storage may also cause performance bottleneck, as all data collection and access have to be done at the sink nodes. Therefore, to provide robust data retrievability, sensed data should be distributed over the whole network instead of employing centralized storage on sink nodes.

Recently, the technique of distributed storage coding has been widely used to disseminate, store and retrieve sensed data. Existing works usually use erasure codes like Reed-Solomon code and Fountain codes, where each storage node stores a linear combination of a subset of source packets sensed by source nodes independently. A collector (e.g., mobile station) can reconstruct all the source packets with high probability by querying any fixed-size subset of storage nodes and solving the linear equations. Dimakis *et al.* [1] proposes a decentralized coding algorithm based on Reed-Solomon code, but the use of Maximum-Likelihood decoding algorithm makes the decoding process inefficient. Subsequently, several distributed algorithms [2]–[5] have been proposed to take advantage of the linear coding property of Fountain codes. Most of these schemes suffer from the drawback of decoding all packets or nothing. If the collector is interested in partial data information in a large-scale network, the recovery of data of interest always requires decoding of all source packets, which requires additional cost on decoding unnecessary data. Although there has been many works to address the partial data recovery problem [6]–[9], the collector still cannot flexibly specify the subset of source nodes of its interest and recover corresponding partial data. Consider a mission-critical scenario where sensors are deployed in a large-scale battlefield to sense the environment. At a certain time, the commander is willing to obtain the environmental information in a specific subarea in order to make a tactical plan, and thus only data sensed by source nodes in the target area should be retrieved. Moreover, many existing works utilize the random walk to disseminate data from source nodes to storage nodes. Traditional random walk algorithm is inefficient in this scenario since only one storage node is supposed to store the data during one random walk, which means the source node should launch a large number of random walks to disseminate sensed data to the distributed storage nodes.

In this paper, we propose a distributed storage coding algorithm in wireless sensor networks, which ensures the efficient dissemination of source packets to storage nodes and flexible recovery of data measured by any given subset of source nodes of interest. Our scheme improves the efficiency of traditional random walk algorithm, based on a variant of Metropolis Algorithm, to disseminate the sensed data. More specifically, by taking advantage of broadcast nature of wireless transmission, we reduce the forwarding times of random walk required by source nodes for distributing source packets, and thus greatly reduce the communication cost in data dissemination procedure. In addition, to enable the collector to selectively retrieve data generated by any subset of source nodes of interest, each storage node encodes all the source packets received from the same source node into one packet so as to create a distributed Fountain code for all source nodes. Subsequently, at the end of any t-th time slot, a mobile collector can move randomly in the sensor network and query

978-1-4244-6404-3/10/$26.00 ©2010 IEEE

any $t(1+\epsilon)$ storage nodes to recover the data generated by the target source nodes.

The remainder of this paper is organized as follows. In Section II, we introduce the network model and give an overview of Fountain codes and random walk. Section III describes our distributed storage coding algorithm including efficient data dissemination and flexible data recovery. We provide simulation results in Section IV, and conclude the paper in Section V.

II. Network Model and Preliminaries

A. Network Model

We formulate a wireless sensor network of n sensor nodes as a random geometric graph [10], denoted as $G(n,r)$. Each pair of two nodes could directly communicate if their distance is not larger than the radio range r. We can assume, without of loss of generality, that all the n nodes in the network are equipped with limited memory so they all can be used as storage nodes. Besides, only a subset of sensor nodes with size of k are equipped with some sensing device, and then those nodes are referred to as source nodes. The sensing time of each source node is divided into K time slots with identical slot length, and each source node produces one packet per time slot. With respect to the timing setting, we adopt the same asynchronous time model as that used in [11].

B. Preliminaries

1) Fountain Codes: Fountain codes have a typical property that the encoding procedure can be performed repeatedly to generate unlimited number of coded packets. The encoded packet is generated by conducting bitwise XOR operation on a subset of source packets, and all the source packets can be decoded from any subset of the encoding packets whose size is slightly greater than the number of source packets. Moreover, its online encoding and decoding procedures could be accomplished with low computation cost.

LT codes [12], the first realization of Fountain codes, can recover K original source packets from any K+$O(\ln^2(K/\delta))$ coded packets with probability 1-δ. The decoding procedure is performed by the efficient Belief Propagation decoder [13] with complexity $O(K\ln(K/\delta))$. As an important parameter in Fountain codes, code degree d is defined as the number of source packets that are combined into one coded packet. In LT codes, the distribution of code degree is defined by Ideal Soliton distribution or Robust Soliton distribution. The Ideal Soliton distribution is $\rho(i)$, i.e., $P\{d = i\}$, where $\sum_{i=1}^{K}\rho(i) = 1$ and

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

The Robust Soliton distribution is $\mu(i)$, where $\mu(i) = (\rho(i) + \tau(i))/\beta$ and $\beta = \sum_{i=1}^{K}\rho(i) + \tau(i)$. Let $R = c\cdot\ln(K/\delta)\sqrt{K}$ for a suitable constant c, and define $\tau(i)$ as follows,

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

To generate n coded packets, every source packet is combined in average C_0 coded packets where $C_0 = n\sum_{d=1}^{K} d\mu(d)/K$.

2) Random Walk: To achieve the distribution of code degree in Fountain codes, random walk has been used in several decentralized coding mechanisms [2]–[4]. If the distribution of a random walk with sufficient length L_0 stopping at a node is referred to a time-reversible Markov chain [5] with a nonuniform stationary distribution based on the node's code degree, the Metropolis algorithm [14] can be well employed because it is a common alternative to construct a time-reversible Markov chain with a specified stationary distribution from any irreducible Markov chain on a state space Ω. Consider a stationary distribution $\pi_x = d(x)/D$, where $d(x)$ is the code degree for any node $x \in \Omega$ and $D = \Sigma_{x\in\Omega}d(x)$. We can use the Metropolis algorithm to construct a Markov chain with the designed stationary distribution via the proper formulation of transition matrix as

$$P_{x,y} = \begin{cases} min(1, \pi_y/\pi_x)/M & \text{if } x \neq y \text{ and } y \in N(x) \\ 0 & \text{if } x \neq y \text{ and } y \notin N(x) \\ 1 - \sum_{y\neq x} P_{x,y} & \text{if } x = y, \end{cases}$$

where $M = max_{x\in\Omega}|N(x)|$ and $N(x)$ is the neighbor set of node x. Moreover, the ratio of neighboring nodes' code degree π_y/π_x can be reduced as $d(y)/d(x)$ without knowing the global information D, and thus this Metropolis algorithm can be utilized in the distributed storage scenario.

III. Distributed Storage Coding In Wireless Sensor Networks

During the sensing procedure, every source node generates one packet per time slot. Consider the urgency of the sensed data in a mission-critical scenario, the packet should be disseminated immediately so that the storage node can perform encoding on the fly. In this section, we first describe a data dissemination scheme which aims to improve the efficiency of traditional random walk algorithm to deploy the distributed LT codes over storage nodes. Then, a detailed description of our proposed distributed storage coding scheme will be provided.

A. Efficient Dissemination of Source Data

After packet generation, to distribute data packets to storage nodes, several random walks are launched by source nodes, move forward the packets for sufficient steps, and finally stop at some storage nodes. The communication cost in this process is mainly determined by two factors: the length of one random walk, i.e., L_0, and the launching times of random walk. Since sufficient transmission steps are needed for a random walk to ensure that the code degree can converge to a stationary distribution, we are not expected to reduce the length of random walk to a large degree. However, by employing the broadcast nature of wireless transmission, we can reduce the repetition times of random work to improve the communication efficiency.

Consider the ideal case where no duplicate source packet arrives at the same storage node, one storage node x should receive $d(x)$ packets from each source node over K time slots. Therefore, n storage nodes will receive $nk\sum_{d=1}^{K} d\mu(d)$

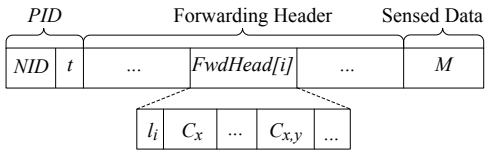


Fig. 1: Data structure of source packet

packets from the k source nodes totally, that is, one source node should deliver C_0 copies for every packet in each time slot. If one random walk only transmits one source packet to one storage node, every source node should continuously launch C_0 times of random walks for delivering each source packet to C_0 storage nodes, and thus the communication cost to disseminate one packet is proportional to L_0C_0. Considering the big value of n in a large-scale sensor network, the communication overhead may be significantly large especially for the source nodes and their neighboring nodes.

The intuition behind our proposed design is as follows: in the traditional random walk algorithm for data delivery, one node may cache many copies of one identical packet corresponding to different random walks in its buffer, and forward one copy to only one of its neighboring nodes selected by the Metropolis algorithm in each time of transmission. In wireless sensor networks, due to the broadcast nature, when one node forwards a packet to one neighbor node, this packet can also be received by all other neighbor nodes within the transmission range. Accordingly, if one packet is buffered on one node for more than one copy, instead of independently forwarding all these copies to different neighbor nodes through many times of transmission, the node can broadcast the packet to all neighbor nodes through only one time of transmission.

To meet our design needs, we propose a new data structure for source packet D to encapsulate the actual sensed data M (see Fig. 1), which is designed to enjoy the broadcast nature of wireless transmission. In our design, every source packet is identified by a packet ID (PID), which is a combination of source node's ID (NID) and the sequence number of the time slot when this source packet is measured, i.e., t. In the forwarding header $FwdHead$ for multiple random walks, each forwarding unit $FwdHead[i]$ consists of the number of steps left (l_i) and the forwarding times for one random walk ($C_{x,y}$). Assume that the node x is supposed to forward one data packet for C_x times. The expected times of forwarding this packet to its neighbor node y, denoted by $C_{x,y}$, can be calculated by a variant of Metropolis algorithm as follows,

$$C_{x,y} = \begin{cases} C_x P_{x,y} & \text{if } x \neq y \text{ and } y \in N(x) \\ 0 & \text{if } x \neq y \text{ and } y \notin N(x) \\ C_x - \sum_{y \neq x} C_{x,y} & \text{if } x = y. \end{cases} \quad (1)$$

With the employment of this algorithm, the stationary distribution of random walk is same as that using original Metropolis algorithm. The expected value $C_{x,y}$ is equal to the number of packets forwarded to the node y when its neighbor x is expected to forward C_x packets totally. Obviously, by broadcasting such one packet to all neighbor nodes, the node x can reduce its communication cost to almost $1/\sum_i C_x$ since the forwarding header is much smaller than the sensed data. This significant advantage could definitely reduce the energy consumption which is an important consideration for sensor nodes during the data dissemination stages.

Algorithm 1 Packet Forwarding Mechanism

```
1: procedure PACKETFORWARDING(D_[PID], x, z)
2:     for i ← 1 to |FwdHead| do
3:         if C_{x,z} = 0 then
4:             Delete FwdHead[i] from D_[PID];
5:         else
6:             l_i = l_i − 1;
7:             if l_i = 0 then
8:                 Encode D_[PID] with existed coded packet;
9:                 Delete FwdHead[i] from D_[PID];
10:            else
11:                C_z = C_{x,z};
12:                Compute C_{z,y} for each y ∈ N(z);
13:    if FwdHead is empty now then
14:        Drop D_[PID];
15:    else
16:        if The buffer on y contains D'_[PID] then
17:            Merge FwdHead into D'_[PID];
18:        else
19:            Store D_[PID] in the buffer;
20: end procedure
```

B. Our Proposed Distributed Storage Coding Scheme

In this section, we present the detailed scheme for achieving distributed data storage coding. Assume there are n storage nodes and k source nodes in a wireless sensor network, we now list activities in different phases:

1) Initialization: Due to the limited storage capacity, storage nodes cannot store the original source packets. Before deployment of the wireless sensor network, every storage node x should determine the quantity of packets combined into one packet, i.e., code degree $d(x)$. This parameter is computed by every storage node itself following the distribution $\mu(i)$ in LT codes. To compute the transition probabilities, every storage node exchanges its code degree with all neighbor nodes.

2) Data Generation: At the t-th time slot, every source node x independently generates one data packet M containing the sensed information and then attaches the packet ID and a new forwarding header. This new forwarding header consists of only one forwarding unit, where l_1 and C_x are initialized as L_0 and C_0, respectively. By computing $C_{x,y}$ according to (1), the fresh source packet $D_{[PID]}$ is ready for delivery. Note that two source packets contain the same data packet M if they have the same packet ID.

3) Data Dissemination: After source nodes broadcast source packets to their neighbor nodes, all the storage nodes could use our proposed forwarding mechanism as depicted in Algorithm 1 to process the received packets $D_{[PID]}$. When a storage node z receives a source packet broadcasted by its neighbor node x, it first checks the forwarding times in every forwarding unit. If any $C_{x,z}$ equals to 0, this forwarding unit is removed from the header. Especially, this packet should be

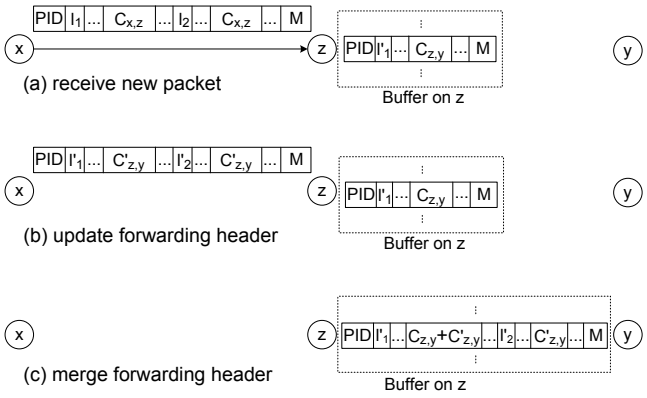


Fig. 2: Packet merging procedure

dropped if there is no forwarding unit left after the checking procedure, in other words, node z is not expected to encode or further forward this packet. Then, l_i is reduced by 1 as $l_i^{'}$ in every forwarding unit. Note if the updated $l_i^{'}$ is equal to 0, which means a stop sign for the corresponding random walk, this packet should be encoded by the storage node and the forwarding unit is removed from the forwarding header. Similarly, this packet will not be further broadcast if the forwarding header is empty after removal of forwarding units. Otherwise, C_z is set to $C_{x,z}$, and later is used to calculate the forwarding times $C_{z,y}$ for each neighbor node y according to (1). Particularly, the data packet cannot be encoded if current storage node has already encoded this packet or the current code degree has reached its expected value computed in initialization phase. In such case, a new random walk should be launched such that the $l_i^{'}$ is set as the initialization value L. Once all the forwarding units are updated, the storage node searches the source packet with same packet ID in its buffer, and then merges two forwarding headers into one.

A simplified packet merging procedure is shown in Fig. 2. We assume that, in the t-th times slot, the storage node z receives a source packet from its neighbor node y and caches it in the buffer. Before this packet is broadcast to neighbor nodes, z receives a source packet from another neighbor node x where the packet ID is same as that in the cached packet. After updating the forwarding header of the new arrived packet, one forwarding unit could be accumulated with another unit in the cached packet if they have the same $l_i^{'}$, and all other forwarding units are simply inserted into the cached packet.

4) Data Encoding: Every coded packet in the storage node consists of the source node ID(NID), the coded sensed data M' and the t of source packets combined in this coded packet. Once one random walk is stopping at a storage node, the corresponding source packet is then combined with the existing coded packet if these two packets consist of the same NID. According to the encoding algorithm in LT codes, the combination procedure is performed by the efficient bitwise XOR operation on the original sensed data M in the new arrived source packet with the coded sensed data M' in existing coded packet. Therefore, each coded packet with NID in the storage node in t-th time slot is a combination of a subset of t source packets which are generated by the source node with NID over the past t times slots.

5) Data Querying and Decoding: Whenever a mobile collector is interested in the status of some specific area, it can recover all the data in the subarea through querying any $t(1+\epsilon)$ storage nodes where t is the current time slot and ϵ is defined as recovery overhead. Specifically, in the querying procedure, the collector specifies the NIDs of source nodes deployed in that subarea, and every queried storage node returns a coded packet with respect to every specified NID if available. Therefore, the collector can access up to $t(1+\epsilon)$ coded packets for every specified NID, each of which is a combination of a subset of t source packets generated by the source node with the NID in that area over the past t times slots. The size of the subset is same as the current code degree of the queried storage node. Subsequently, Belief Propagation decoder is utilized to decode all the t source packets of every specified source node.

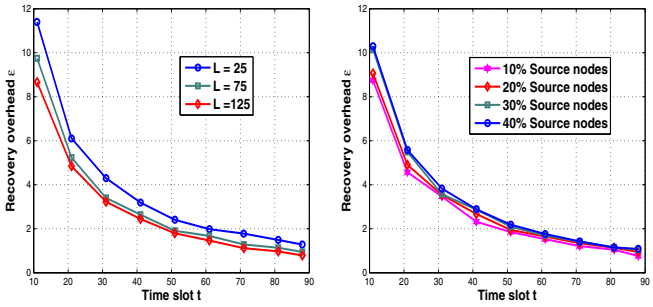
IV. Scheme Evaluation

In this section, we evaluate the flexibility and efficiency of our proposed scheme through the C-based simulator. We consider a wireless sensor network $G(n, r)$ consisting of 1000 nodes, 100 of which have the sensing function, i.e., $n = 1000$, $k = 100$. Besides, the radio range r is set to be 4 so that each node has four neighbors averagely, and the sensing time of each source node is divided into 88 time slots, i.e., $K = 88$. In each time slot, the simulator first disseminates new source packets from every source node to storage nodes, and then performs the data querying and decoding. All the results are obtained based on an average of 50 runs.

A. Recovery Overhead

To recover source packets generated by any subset of source nodes over the past t time slots, a mobile collector can query any $t(1 + \epsilon)$ storage nodes where ϵ is the recovery overhead. Affected by the length of random walk L_0 and the time slot t, the recovery overhead indicates the robustness of data storage. At the end of K-th time slot, the actual code degree distribution in the storage nodes is expected to be very close to the Robust Soliton distribution $\mu(i)$ in LT codes. According to LT codes, K source packets can be recovered from any K+$O(\ln^2(K/\delta))$ coded packets, and therefore the recovery overhead ϵ is close to $O(\ln^2(K/\delta)/K)$. Before the K-th time slot, the recovery overhead should be larger because the distribution of actual code degree is more or less different with the expected Robust Soliton distribution.

Fig. 3(a) depicts the recovery overhead as a function of both time and the length of random walk, where a mobile collector is recovering source packets generated by 20 percent of source nodes. In the first several time slots, the recovery overhead is 5 or 6 times larger than that in the later time slots since the number of source packets at the beginning is not large enough to cover all storage nodes. Actually, our scheme could recover $0.2kt$ instead of t source packets through querying $t(1+\epsilon)$ storage nodes, and therefore the meaning of recovery



(a) (b)

Fig. 3: Recovery overhead. $n = 1000, k = 100, K = 88$. (a) different length of random walk, (b) recovering data generated by different fraction of source nodes with $L = 75$.

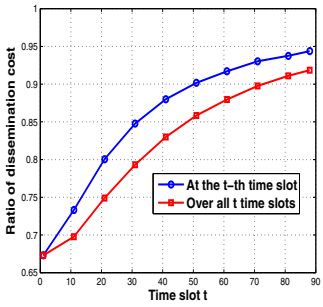


Fig. 4: Ratio of dissemination cost of our algorithm to that of traditional random walk.$n = 1000, k = 100, K = 88, L = 75$.

overhead here is slightly different from that in related works. As time moves forward, more source packets are generated to be disseminated so that the actual code degree in the storage nodes is asymptotic to the Robust Soliton distribution. Once the actual distribution is more similar to the Robust Soliton distribution, the recovery overhead can be reduced to a large extent. Moreover, the actual stationary distribution of random walk is more asymptotic to the expected distribution when the length of random walk is longer. Consequently, the recovery overhead decreases if the length of random walk increases.

As shown in Fig. 3(b), the recovery overhead is not affected very much by the fraction of source nodes whose packets are to be recovered. This is because our proposed algorithm is to encode source packets from same source node. Enjoying this property, the mobile collector could recover source packets of any given subset of source nodes while querying similar number of storage nodes.

B. Dissemination Cost

Utilizing the broadcast nature of wireless transmission, our proposed packet forwarding algorithm saves a large part of communication cost during the data dissemination stage in every time slot. Fig. 4 depicts the ratio of dissemination cost of our algorithm to that of traditional random walk algorithm. When time moves forward, the actual code degree is close to its expected value, and then more and more new random walks are launched by the storage nodes due to the limitation of code degree. The sum of forwarding times $\sum_i C_{[l_i]x}$ is usually small in the new launched random walk, and therefore the advancement of our algorithm is reduced to some extent. Although the ratio increases a lot in the later time slots as shown in Fig. 4, the overall ratio is still considerable, which is smaller than 90 percent through almost all time slots.

V. Conclusion

This paper is to investigate the robustness of data storage and efficiency of data retrieval in wireless sensor network. Our proposed distributed storage coding enables a mobile collector to recover data of any subset of source nodes at any time via querying any $t(1+\epsilon)$ storage nodes. By improving the random walk algorithm, the source packet is designed such that the data dissemination could enjoy the broadcast nature of wireless transmission while achieving similar stationary distribution. Furthermore, our simulation results show that our proposed coding algorithm does not introduce much more overhead to recover data of larger subset of source nodes. This is particular useful in the large-scale network when we are only interested in the status of subarea instead of the whole area.

VI. Acknowledgements

This work was supported in part by the US National Science Foundation under grants CNS-0716306, CNS- 0831628, CNS-0746977, and CNS-0831963.

References

[1] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes," in *IPSN*, April 2005, pp. 111–117.
[2] S. A. Aly, Z. Kong, and E. Soljanin, "Raptor codes based distributed storage algorithms for wireless sensor networks," in *ISIT*, July 2008, pp. 2051–2055.
[3] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Distributed fountain codes for networked storage," in *ICASSP*, vol. 5, May 2006.
[4] S. A. Aly, Z. Kong, and E. Soljanin, "Fountain codes based distributed storage algorithms for large-scale wireless sensor networks," in *IPSN*, 2008, pp. 171–182.
[5] Y. Lin, B. Liang, and B. Li, "Data persistence in large-scale sensor networks with decentralized fountain codes," in *INFOCOM*, May 2007, pp. 1658–1666.
[6] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: Maximizing sensor network data persistence," in *ACM SIGCOMM*, September 2006, pp. 255–266.
[7] W. Wang and K. Ramchandran, "Random distributed multiresolution representations with significance querying," in *IPSN 2006*, pp. 102–108.
[8] Y. Lin, B. Li, and B. Liang, "Differentiated data persistence with priority random linear codes," in *ICDCS*, June 2007, pp. 47–47.
[9] D. Wang, Q. Zhang, and J. Liu, "Partial network coding: Concept, performance, and application for continuous data collection in sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, no. 3, pp. 1–22, 2008.
[10] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *ITIT*, vol. 46, no. 2, pp. 388–404, 2000.
[11] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: design, analysis and applications," in *INFOCOM*, 2005, pp. 1653–1664.
[12] M. Luby, "Lt codes," in *FoCS*, 2002, pp. 271–280.
[13] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *ITIT*, no. 2, pp. 569–584, 2001.
[14] M. Mitzenmacher and E. Upfal, *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, January 2005.