Enhancing Reliability in Storage-Centric Sensor Networks

Amir Nayyeri and Tarek Abdelzaher
Department of Computer Science
University of Illinois at Urbana-Champaign
{nayyeri2, zaher}@cs.uiuc.edu

***Abstract*—In this paper we address the reliability problem in storage-centric sensor networks deployed in hazardous environments. We use the nodes' extra flash memory to save distributed encoded blocks of data. We employ erasure coding to substantially improve the trade-off between storage reliability and required disk capacity. Moreover, efficient mechanisms are designed for spreading the encoded pieces over the network. Mathematical and experimental results show that erasure coding preserves the data from both crashes and memory overflow, substantially outperforming traditional schemes.**

I. Introduction

This paper presents a reliable storage system for sensor networks that uses erasure coding to significantly improve the trade-off between storage reliability and required storage capacity. The scheme is especially suitable for networks that operate disconnected in harsh environments for long periods of time, where data cannot be uploaded immediately.

Several sensor networks have been deployed for different environmental monitoring purposes. In many of these networks, nodes are responsible for sensing and transmiting data through single or multi hop paths to a central base station to be saved and processed when necessary. Consequently, the main challenge in such systems is to minimize communication energy consumption. In contrast, in many systems it is not vital to propagate data to the base station fast. This observation inspires thinking of the sensor network as a storage system. Nodes are equipped with a local disk, and the collaborative goal is to record and keep as much data as possible. The network may, in fact, operate disconnected with only rare opportunities to upload data to a central station. The main challenge in this new setting shifts from communication energy optimization to reliable storage. There exist some results for partially connected networks (e.g. [9], [10]), as examples of storage-centric networks. However, most of them work with low-bandwidth data. Recently, [1] considered deploying collaborative acoustic recording sensors, where the nodes have to deal with larger amounts of data. This paper considers a similar problem in a different environment, where crashing is probable due to natural events.

Generally, there are two important resources that should be considered in any design for storage-centric networks: disk and energy. Contrary to many typical applications of sensor networks, storage is the bottleneck in sensor networks that operate primarily in disconnected mode. This is because the total time that a single node spends on communication is insignificant compared to the deployment lifetime of the network. For example, a recording embedded PC system equipped with solid state storage and a WiFi radio (54 Mbps), nominally requires only a few minutes to upload 1GB to another node. Considering the ratio between radio and CPU power consumption, a few extra minutes of communication will mean several fewer minutes of lifetime; not something to worry about when deloyment lifetime is measured in days or weeks. Even a larger solid state device (e.g., 10GB) should be uploaded fast enough. For a node deployed to record data for a long duration (e.g. 30 days) depleting energy, say, two hours sooner means losing less than 0.003 percent of the data, which is negligible. For a similar example see [1]. In short, it is the solid state device capacity and not energy that is the bottleneck. Incidentally, most next generation motes, such as Imote2 and mPlatform have a WiFi interface.

The main consideration of this paper is how to take advantage of the extra disk space to enhance reliability in sensor networks. Besides using simple replication of blocks, we introduce the novel idea of using erasure coding to achieve a significantly higher reliability with a same storage requirement. Since erasure coding partitions each block into smaller pieces, it also contributes amazingly to the load balancing of the system. So, the advantage of erasure coding over simple replication is two-fold. As a result, it does better in different simulated experiments.

The other problem we address in this paper is the strategy to scatter the coded blocks all over the network. Two randomized methods are suggested. The first one is completely local (each node only needs to know its neighbors), while the second one requires a more global view, but works better for sparse networks. Finally, we analyze the effectiveness of different schemes mathematically, and through simulation.

The rest of this paper is organized as follows. In the next section we give a brief overview of erasure coding. Section 3 is devoted to explanation of our algorithms and protocols. In Section 4 and 5 we analyze the methods mathematically and experimentally, respectively. Section 6 introduces the related work. Finally, Section 7 concludes the paper.

II. Erasure Coding Background

Generally, Erasure codes produce a large set (of size n) of encoded blocks from a smaller set of input blocks (of size

k), so that the original data is recoverable having a sufficient amount ($(1+\epsilon)k$) of encoded data. The rate of such a coding is defined as $r = n/k$. The values of ϵ and maximum possible r are different in each particular Erasure coding method. Ideal digital fountain codes are defined as those with maximum r and minimum ϵ. That is, infinitely many coded blocks can be generated such that the original data are retrievable from any k of them [2]. Actually, the first condition is not that important for our application, as we do not need to produce a lot of coded blocks. However, the value of ϵ affects the performance of our methods significantly. These conditions make the Reed-Solomon (RS) [3] algorithm a proper candidate for our work, while they may also work with other proposed erasure coding techniques.

The main idea behind RS is representing the data as a polynomial. It is well know that it is possible to reconstruct a polynomial of degree at most k from k points. So, thinking of the k blocks as a polynomial of degree k, the sender evaluates it in n different points, and transmits the results to the receiver. The receiver is capable of reconstructing the main message if it receives at least k blocks. For more detailed explanation we refer the reader to [3].

III. Using Redundancy to Improve Reliability

Nodes are vulnerable to damage or loss in many applications of sensor networks. Therefore, it is helpful to have fault-tolerant schemes to preserve data in such hazardous places. In a typical storage-centric sensor network, nodes are responsible for collecting and keeping data on their local disks. Failure of a single node will result in loss of all its data. Obviously, spreading copies of each segment over the network alleviates this situation. It costs overhead in terms of energy and extra storage to improve the reliability of the system. Cheaper methods are better. Particularly, (as explained in the introduction) we are interested in schemes that minimize storage overhead.

We call a system deterministically α-$reliable$ if it tolerates failure of α nodes. In other words, it is possible to retrieve all of the collected data, from an α-reliable sensor network if no more than α nodes are crashed. In addition, we define probabilistic reliability of a system as the expected ratio of the overall data it eventually collects successfully.

Designing a reliable protocol is achieved through solving two subproblems. The first one is how to make redundant data enhance the reliability of the system, which is going to be addressed in each sensor separately. The second is how to spread the data all over the network, which needs distributed decision making. The following subsections address these two problems.

A. Replicating Schemes

Simple Replication (SR) is the most primitive approach we applied. To achieve an α-reliable system, $SR(\alpha)$ sends α extra copies of each block to other nodes (The problem of finding α distinct nodes is addressed in the next subsection). After the copies are made, $\alpha + 1$ blocks are settled in distinct nodes, which makes the original data accessible in the presence of up to α failures.

SR can be generalized by applying any randomized decision making alogrithm for the number of replications of a single block whose expected value is α. One example is always generating $\lfloor\alpha\rfloor$ blocks, and generate one extra block with probability $\alpha - \lfloor\alpha\rfloor$.

SR is favorable for its simplicity. However, it has significant shortcomings, which makes it far from a perfect method. The most serious disadvantage it brings is its extremely high storage requirements. To make an α reliable system, SR requires $\alpha + 1$ times the basic storage requirement. This inspires using more complicated encoding methods to achieve similar efficiency with lower resource requirements.

Erasure Replicating (ER) is motivated by the coding theory (see section 2). We call an ER of degrees k_1 and k_2, denoted $ER(k_1, k_2)$, if it produces k_2 encoded blocks from k_1 input blocks. Thus, for an α-reliable system we need $ER(k, k+\alpha)$, where k is a fixed constant in the ER algorithm. This method is applicable if there exists α/k times the basic storage of extra disk space. This means that by selecting higher values for k we may reduce the storage requirements while keeping the deterministic reliability degree fixed, as long as the value of $k + \alpha \leq n$.

In fact, the values of k and α affect the outcome of the algorithm in different ways. With a higher α we expect a higher reliability and energy consumption, since we are sending more data all over the network. The affect of k, on the other hand, is not that easy to analyze. We have to send more pieces of data all over the network for a larger value of k. Although the overall data size does not change substantially, it is harder to find adequate distinct receivers. On the other hand, a larger value of k means smaller pieces of data which may result in a more balanced network and less overflow.

B. Spreading Schemes

In both SR and ER, coded blocks should be scattered throughout the network. Ideally, adequate receivers should be found for each node, such that storage usage is balanced and the energy consumption for transferring the blocks is minimized.

By loosing the condition of energy optimality we get a more tractable problem, whose solution is almost as effective as the optimal solution as the storage is the bottleneck for our system. Finding receivers all over the network is a vital task, in that its failure undermines the effect of the replication scheme. Moreover, it may improve distributed disk usage significantly, by spreading the data chunks in a balanced way. This may reduce the instances of data overflow for the disks which is another important root cause of missing data besides node crashes.

We introduce two different protocols to find receivers. We call the first Probabilistic Spreading (PS) and the second Fixed Spreading (FS).

PS is an algorithm based on random walks. Intuitively, for spreading $K = k + \alpha - 1$ blocks, the node initiates K

random walks originating at itself and ending in distinct nodes. More precisely, for each k blocks, the source node generates $k + \alpha$ blocks and gives them some common set ID, s (then increments s for the next set). It stores one of the blocks in local storage and sends the others to random neighbors. Receiving a block, if the neighbor has one with the same set ID, it forwards the others to some random neighbors. To make the algorithm more efficient, nodes write their identifiers on the block as it is passed to avoid loops.

As explained previously, the spreading scheme affects load-balancing. We add a probabilistic decision to this method to make it more flexible for different kinds of networks. Namely, each node forwards a block passed to it with probability fp (forwarding probability), even if it does not have same set block in its memory ($fp = 0$ is the regular algorithm). Larger values for fp will result in a network where blocks are scattered in a wider region, where we expect better load balancing. Observe that, very high values cause nodes to forward a block until it gets into a trap (where it cannot be forwarded to a new node). This is obviously disadvantageous ,wasting the network energy and also putting many same set blocks on a single node that decreases the reliability. So, determining a proper fp is the tricky job.

In PS each node needs to know only about its immediate neighbors, so the decisions are made very locally. Intuitively, we expect this algorithm to work well on dense graphs, while it may get into trouble when the underlying graph is sparse. In fact, in sparse graphs, particularly when K is not a small number, finding that number of distinct receivers completely though random walks may not work well. The next protocol achieves better results for sparser networks by trading efficient allocation and complexity.

FS is also a parameter tunable algorithm. Its parameter is m, that is the locality of the receivers of nodes. More precisely, in $FS(m)$ each node tries to find its receivers in its m-neighborhood. In this method, each node broadcasts a hello message with $TTL = m$ after the network has been deployed. Each sensor will receive the messages of its m-neighborhood. It keep tracks of these nodes and selects K of them randomly whenever it decides to send extra blocks. Of course, we should be aware that some nodes may fail in the middle. To resolve this problem, nodes send ping messages to their immediate neighbors. Whenever a node discovers that one of its neighbors is crashed, it will broadcast a dying alert message with $TTL = m + 1$. Since many nodes may transfer ping messages with a single node, each one of them waits a random time before sending the alert message, and if it hears the same message sent by another fellow, it will not re-send it. The ping messages are exchanged very infrequently, and they have a very low overhead compared to data uploads. At first glance, FS may look like a rather complicated protocol, which may bring large overhead in terms of energy. However, observe that the blocks passed among the nodes are very large compared to the messages required for neighbor discoveries and dying alerts. Furthermore, we can exchange these messages very infrequently, as nodes have to make decisions once in a while for transferring blocks. Hence, a small saving in block communications offsets overhead. Similar to fp in PS, proper selection of m is definitely important in FS. Determining m is a trade-off between achieving higher reliability and higher energy efficiency. Larger values of m can support larger values of k, while they may oblige a node to send its extra blocks far from itself, which requires higher energy consumption. Also, observe that higher values of m may contribute to load balancing by spreading the data over a larger subset of nodes.

IV. Analysis

A. ER vs. SR, Probabilistic Analysis

Previously, we introduced the deterministic notion of $\alpha-$reliability from the number of crashes a system can tolerate. From deterministic viewpoint ER obviously performs much better than SR, given equal extra storage space. To achieve deterministic α-reliability in simple coding we need at least $(\alpha - 1)S$ extra storage, where S is the basic storage requirement. Nevertheless, this value can be even less than S for proper selection of k_1 in erasure coding.

We also introduced a probabilistic notion of reliably, which is the expected value of the fraction of data that survives. Suppose that we have a network in which each node may crash sometime during the data gathering process with probability p. A meaningful measure is to compute the probability that a particular block of data will survive at the end of the day, given that it has been collected by a sensor. Note that, by summing up these values for different blocks, we get the expected amount of survived data. Assume that the network is going to collect an amount S of data. Also assume that, overall, there exists $S\beta$ extra storage that the network may use to enhance reliability.

We consider the SR version here in which each node replicates $\lfloor\beta\rfloor$ copies from every single block and adds an extra copy with probability $\beta - \lfloor\beta\rfloor$. On the other hand, assume we can apply $EC(k_1, k_1(1+\beta))$, where $k_1\beta$ is an integer. To make this assumption realistic, we should select an appropriate k_1 in the implementation of the algorithm. Note that, the out-degree of erasure may be bounded by the number of the nodes in the network. Here, to ease the computations, we assume that there are sufficient nodes, which is natural since usually $k_2 < n$.

Let LP_{SR} and LP_{ER} show the probabilities that a particular block of data is lost using SR and ER, respectively. Considering two possible cases; the final extra block is copied or not with probabilities $\{\beta\} = \beta - \lfloor\beta\rfloor$ and $1 - \{\beta\}$, respectively, we get $LP_{SC} = \{\beta\}.p^{\lfloor\beta\rfloor+2} + (1 - \{\beta\})p^{\lfloor\beta\rfloor+1}$. On the other hand, we know more than $k_1\beta$ blocks should be lost to make the original data not retrievable in EC, That is:

$$LP_{ER} = \binom{k_1 + k_1\beta}{1 + k_1\beta} p^{k_1\beta+1}(1-p)^{k_1-1} + ... + p^{k_1+k_1\beta} \quad (1)$$

Since the probability of crashing is usually very small, it is meaningful to compare LP_{SR} and LP_{ER} as p approaches zero. It is apparent from the power of p that $\lim_{p\to\infty} LP_{ER}/LP_{SR} = 0$, which means ER works better in terms of the expected amount of surviving data.

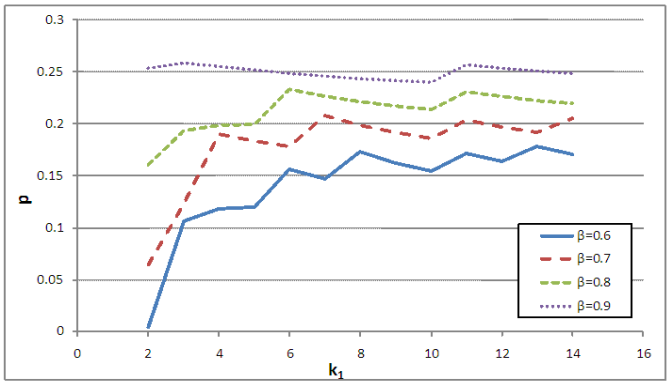

Fig. 1. p vs. k_1 for different values of β

Hence, ER is better for small p. We obtain a sufficient condition to show for how small a p it works. By loosely computing LP_{ER}, we get the following bound from (1).

$$LP_{ER} \leq \binom{k_1 + k_1\beta}{1 + k_1\beta} p^{k_1\beta+1}$$

Also, we get the following trivial bound for LP_{SR}: $LP_{SR} \geq \beta.p^{\lfloor\beta\rfloor+2} + (1-\beta)p^{\lfloor\beta\rfloor+2}$. Thus, the following inequality results in $LP_{ER} \leq LP_{SR}$.

$$\binom{k_1 + k_1\beta}{1 + k_1\beta} p^{k_1\beta+1} \leq p^{\lfloor\beta\rfloor+2}$$

which means:

$$p \leq \sqrt[(k_1\beta - \lfloor\beta\rfloor - 1)]{\frac{1}{\binom{k_1+k_1\beta}{1+k_1\beta}}}$$

for $\lfloor\beta\rfloor + 2 < k_1\beta + 1$, and

$$p \geq \sqrt[(\lfloor\beta\rfloor - k_1\beta + 1)]{\binom{k_1 + k_1\beta}{1 + k_1\beta}}$$

for $\lfloor\beta\rfloor + 2 > k_1\beta + 1$. In practice, we are more interested in the cases where $0 \leq \beta \leq 1$, as we consider storage-constrained systems. Figure 1 shows the required values for different k_1 and β.

B. FS and Load Balancing

Load balancing is another important measure, which affects the reliability. In fact, load balancing lets us utilize resources the best possible way. Here, we analyze our proposed methods in terms of load balancing. Consider FS(m). Let $\iota(u,v)$ be the amount of data v receives from u per time unit. Also, let $\iota(v)$ be the total amount of data v receives per time unit. Without loss of generality, we assume that each node has one unit of extra data to send to the others in the following computation.

$$\iota(v) = \sum_{u \in N_m(v)} \iota(u,v) = \sum_{u \in N_m(v)} \frac{1}{|N_m(u)|}$$
$$\leq \sum_{u \in N_m(v)} \frac{1}{\delta_m} = \frac{deg_m(v)}{\delta_m} \leq \frac{\Delta_m}{\delta_m}$$

where, N_m is the set vertices in an m-neighborhood and $deg_m(v)$ equals the size of $N_m(v)$. Also, δ_m and Δ_m are the minimum and maximum m-degree, respectively. We get a similar lower bound for $\iota(v)$ using the same approach. Finally, we have the following:

$$\frac{\delta_m}{\Delta_m} \leq \frac{deg_m(v)}{\Delta_m} \leq \iota(v) \leq \frac{deg_m(v)}{\delta_m} \leq \frac{\Delta_m}{\delta_m}$$

Since, we expect an almost regular graph from our deployment, these values are supposed to be very close to one, which means we have a good load balancing. However, if the graph is far from regular, the bounds may be arbitrarily bad. In that case, load balancing schemes are vital to alleviate the situation.

C. Dependent Node Crashes

In the computations above, we assumed that node crashes happen independently. Nevertheless, this may not be a very realistic assumption, especially when nodes stop working due to some disaster in the area. Therefore, it is preferred to put data of the same block at locations that are far apart, to make the independence assumption more realistic. Here we consider a simplified model to analyze FS from this perspective.

Consider $FS(m)$, in which a node wants to put k different coded blocks in different parts of the network. We assume that the geographic area that this node will send its coded blocks to is a circle with radius $m\alpha$, for some fixed α which may be related to the transmission ranges of the nodes. Also, we assume that, a disaster only damages those nodes that are closer than β to it. In such a case, if a node crashes and we know that there is no related block stored in a node closer than β to it, the independence assumption is correct. This fact happens with a probability higher than,

$$\left(\frac{\pi(m\alpha)^2 - \pi\beta^2}{\pi(m\alpha)^2}\right)^{(k-1)} = \left(1 - \frac{\beta^2}{(m\alpha)^2}\right)^{(k-1)} \quad (2)$$

Simply, with larger m and smaller k the independence assumption is more realistic.

V. Evaluation

In this section, we analyze the performance of our protocols through simulation. Specifically, we study the overall data miss ratio and energy consumption under various network resource conditions. Here, we focus on evaluating the systems in terms of probabilistic reliability. From the explanation in the previous chapter, the reader may observe that in the deterministic notion of reliability, the erasure method is clearly better than simple replication. All the experiments run under the setup explained in the next subsection.

A. Simulation Setup

Our experiments were carried out using a customized simulator with an experimental setup adopted from [1], which is a sensor network deployed for distributed acoustic data recording. In all experiments of this section the parameters are set as follows, unless otherwise specified. 100 nodes are deployed over a 100×100 area with uniform distribution. Node transmission range is 15 units. Natural acoustic events

occur with uniform distribution all over the area, each can be heard within an expected range of 10 units. Each node has a local storage of size 550 MB, but it only uses 500 MB of it as a shared part, and keeps the other 50 MB to handle its local traffic bursts. This helps to alleviate overflows. ER is applied with $k_1 = 12$. Event generation rate is 0.6 and crash probability is 0.05.

The discrete time simulator simulates 50000 minutes that is about 35 days. The simulated natural sounds are randomly generated all over the field such that we expect 10 minutes of overall recorded sound in each minute of simulation. We assumed one MB of memory is required to record 10 minutes of sound. Hence, at most 5×10^4MB of data is generated, which ideally can be saved in nodes disks without redundancy.

Throughout this chapter we present diagrams for three different coding schemes (i.e. None, Simple, Erasure) and two different spreading schemes (i.e. Probabilistic and Fixed). Conventions SimpleProb, SimpleFixed, ErasureProb, and ErasureFixed are used to indicate the corresponding coding and spreading schemes together. In None no redundancy is applied. In Simple, nodes transmit a copy of each block with the probability proportional to the expected empty space ratio of the disks. ErasureProb uses erasure coding with random walks. In ErasureFixed each node does erasure if it finds enough receivers, otherwise it backs off to Simple. This feasibility is tested for each block and the proper decision is made.

We have applied techniques to encode and distribute blocks of data to enhance the reliability of the system. Therefore, the fraction of the generated data that is collected at the end of the day is the most important value to measure. Formally, we define Collected Data Fraction (CDF) (or correspondingly Missed Data Fraction (MDF)) as the ratio of collected data (missed data) to all data generated by nature. Note that, a crashed node may result in not recording future events, which contributes a large fraction to MDF, since it is impossible to keep the non-recorded data.

Energy is the next meaningful value to measure, which can be estimated by the average amount of forwarded data. They are proportional since energy consumption is dominated by communication. In the subsequent part we provide some measurements of these two values under different conditions, which are determined by other parameters, like the Event Generation Rate (EGR), erasure coding degree (k_1), and crash probability. Note that EGR also presents the usage ratio of disk spaces. The other thing that we are eager to study is the behavior of the spreading methods PS and FS with respect to their parameters fp and m, respectively.

B. PS(FS) vs. $fp(m)$

As explained in the description of the protocols, the parameters fp and m specifically affect the behavior of the algorithms. So, before assessing the algorithms in different environmental situations we are interested in knowing more about the effect of these attributes.

Determining the value of fp is tricky in that a low value may result in blocks clustered close to the sender, while a large value may force a block to keep moving. In fact, the proper value is dependent on the connectivity of the network. Figure 2 demonstrates the behavior of the MDF according to fp, for Simple and Erasure methods. As in Simple we only need to find one receiver, the values do not change much. This tiny difference roots in data lost due to disk overflow rather than node crashes. On the other hand, we see more a noticeable deviation by changing fp in Erasure. Besides changing the number of overloaded nodes, here, by selecting a large value for fp, we may end up not finding a proper receiver after rejecting some potential candidates. This is the reason for the large jump as fp gets close to one. The energy diagram 3 also rises for fp close to one. In fact, for large fp, packets are forwarded until they get into a trap. However, we expect the traps to be special nodes of the graph (e.g. those with lower degrees). So, in this case all data is sent to specific nodes resulting in a vulnerable system to both overflow and crashes. We expect the energy diagram to be proportional to the expected number of forwarded blocks for a specific fp that is ideally $(1-fp)+2fp(1-fp)+3fp^2(1-fp)+... = 1/(1-fp)$, which is confirmed by the diagram.

Similarly, finding a proper value for m is important in FS. Intuitively, this parameter affects the methods in two ways. First, increasingly, it contributes to load balancing and improves both Simple and Erasure methods. Second, Erasure method backs-off to Simple if it cannot find enough receivers, which is more probable for smaller m.

In this experiment we measure MDF and Energy versus m. Figure 4 shows the effect of the m on MDF. For large enough m, the result is almost optimum, where the remaining fraction of missed data is actually non-sensed that is supposed to be recorded by crashed nodes. In Erasure diagram, a stair-like decrease happens when some nodes find the opportunity to do erasure coding in a one step increment of m. The other figure (5) demonstrates the consumed energy for forwarding the packets inside $m-$neighborhood of the nodes. At the beginning, it grows super-linearly. It is easy to see that its growth rate should be almost quadratic, by estimating the hop count with geographic distance. However, after reaching the boundaries of the region, the growth becomes sub-quadratic (even sub-linear), since few nodes will be added in each increment of the m.

C. Comparison

In this subsection, we compare the performance of all the four possible methods and None with respect to MDF and energy. The comparisons are done in different environmental situations determined by three parameters, Crash Probability, EGR and Transmission Range of the nodes.

One important measure for an environment is its hazardousness that is presented by the crash probability parameter. In the previous section, we analyzed this parameter and suggested an upper bound for the value of proper p for ER with respect to k_1 and β. Here, we setup a simulation to give a better understanding of how this value affects the system. Figure 6 shows how MDF changes with crash probability.

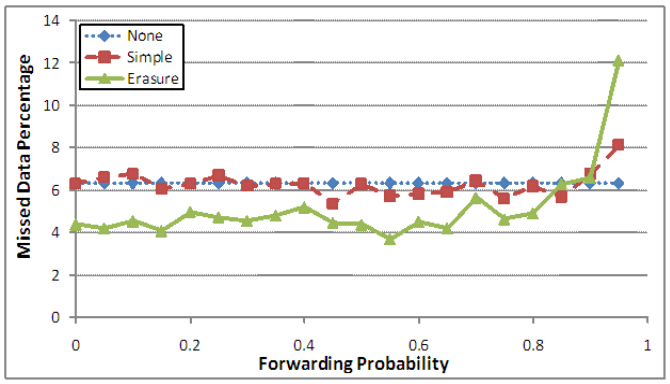

Fig. 2. Missed Data Fraction vs. fp in Probabilistic Spreading

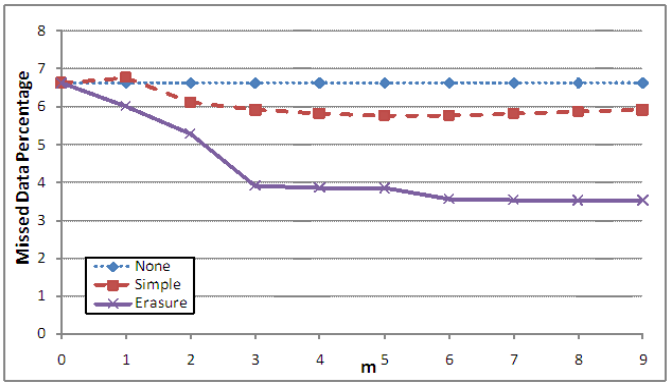

Fig. 4. Missed Data Fraction vs. m in Fixed Spreading

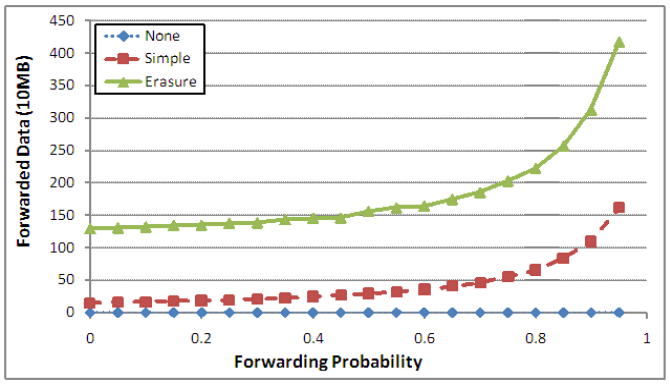

Fig. 3. Energy vs. fp in Probabilistic Spreading

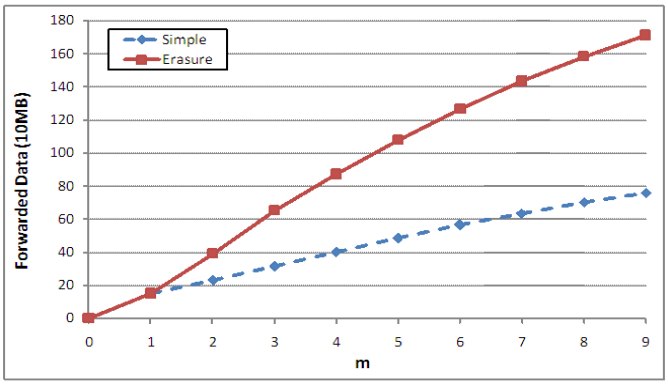

Fig. 5. Energy vs. m in Fixed Spreading

The upper bound of the previous section says that, for $p < 0.18$, erasure coding must be superior, which is confirmed by the experiments (compare ErasureFixed to SimpleFixed). Furthermore, the experiment suggests even a higher upper bound, which means more superiority for erasure coding. Note that, this superiority may come from better load balancing rather than better coding, which is not considered in the theoretical assessment.

The other interesting observation is that the ErasureProb works almost as well as ErasureFixed for small values of crash probability. On the other hand, it degrades for a highly hazardous environment such that it becomes even worse than None for crash probabilities closed to 0.45. The reason is obviously that finding random paths to distinct points is not easy when nodes crashes are likely, and the remaining graph may become disconnected quickly.

Since we apply PS with $fp = 0.55$, we expect that blocks go two hops further from their origin on average. So, the energy consumption for PS should be lower than FS, in which $m = 5$. Figure 7 confirms it for Simple. Surprisingly, the reverse fact happens for Erasure, because after sending some initial blocks, all close neighbors become incapable of accepting later blocks (they come with similar set ID). Consequently, they have to forward packets further spending more energy. This shows that they have to send packet even more than 5 hops on average.

EGR is the next parameter we analyze to figure out how different methods work in networks with different amounts of raw data. Figure 8 presents MDF versus EGR (or equivalently disk usage ratio). The increasing trend in the diagrams is intuitive. However, the rate of increase varies, particularly for ErasureFixed. It even decreases at some points. The reason is that erasure coding can produce an integer number of blocks as output. For example, it produces 15 output blocks for both $EGR = 0.7$ and $EGR = 0.75$. In the latter case the overall data is larger, making MDF smaller by increasing the denominator. For very high EGR, the efficiency of erasure based methods will be dominated by the fact that not enough output blocks can be generated for that fixed k_1. As a result, both ErasureProb and ErasureFixed perform almost the same for $EGR > 0.8$. Another interesting matter in this diagram is that SimpleProb and SimpleFixed are doing worse than None for $EGR > 0.65$ and $EGR > 0.75$, respectively. This tells us that an unwise replication is bad.

In energy diagrams (figure 9), the behavior of ErasureProb and ErasureFixed is not easy to comprehend. Ideally, a node will generate about k_1/EGR blocks and spread all of them but one in the network. For a fixed EGR, each node transmits $k_1 - EGR$ blocks on average, which is almost constant since $EGR \ll k_1$. The other parameter that affects the energy is how far blocks are forwarded. This is fixed for ErasureFixed $m = 5$, so we see a fixed behavior for that curve. In

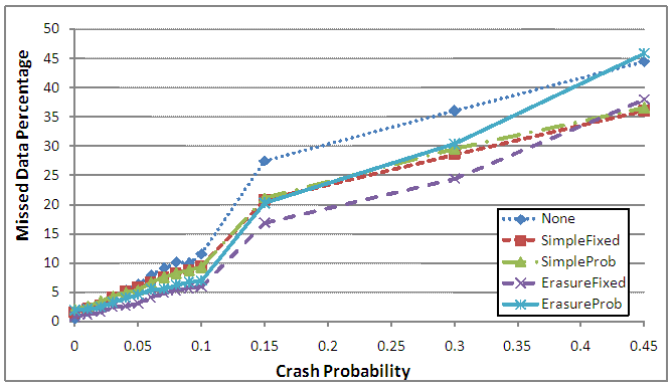

Fig. 6. Missed Data Fraction vs. Crash Probability

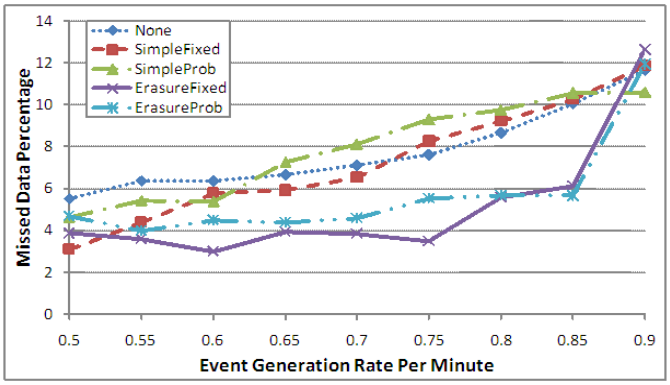

Fig. 8. Missed Data Fraction vs. Event Generation Rate

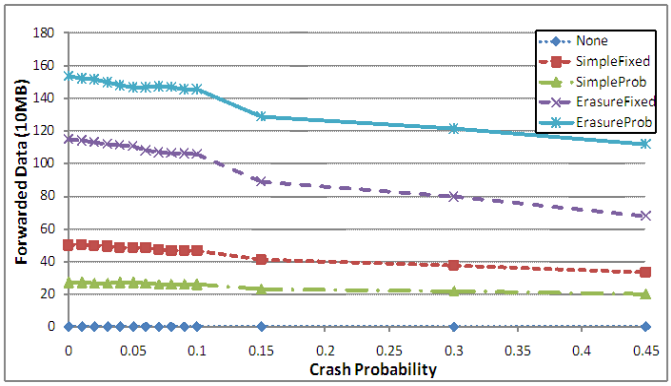

Fig. 7. Energy vs. Crash Probability

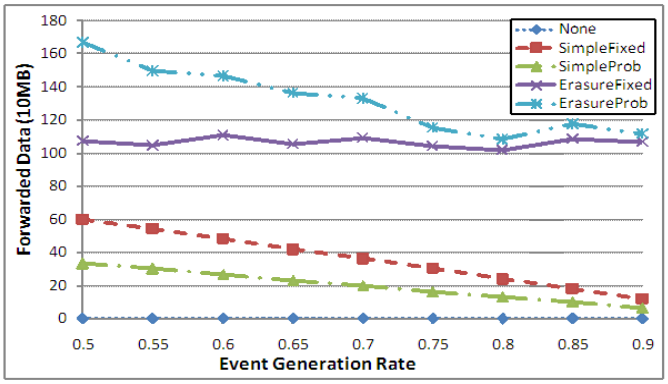

Fig. 9. Energy vs. Event Generation Rate

ErasureProb, for smaller EGR, it produces more output blocks, so it needs to go through longer random walks to find free nodes particularly after some primary blocks are settled down. That is why it spends more energy for smaller EGR.

Since PS works based on pure random walks, it may not work well for sparse graphs, where finding paths to diverse nodes from the same origin is not an easy job. Figure 10 confirms this expectation. However, for dense graphs it works almost as well as FS with the extra payment on energy. For very sparse graphs the erasure methods are unable to find enough receivers, so their performance is poor and they conserve less energy. In the beginning, as the transmission range increases (the graph becomes denser) they perform better, and also consume more energy. Later, consumed energy decreases as range increases, since nodes find receivers with a smaller expected distance (hop count) from themselves. The MDF converges to a fixed point very soon. After that, increasing the ranges only results in decreasing the total amount of forwarded data by reducing the number of neighbor pairs.

D. Overflow vs. Crash

Data loss may happen in our system due to two different reasons: node crashes and memory overflow. So far, we have shown that we can use redundancy to reduce the effect of crashes. Since, in erasure coding, each block is ground to smaller size pieces and the pieces are scattered all over the network randomly, we believe it will lead to a more load balanced network. Of course parameters like m and fp significantly affect the degree of balancing. Figure 12 confirms this guess. In this experiment, we apply the FS method with different coding schemes. We set $m = 5$ for three of the experiments and $m = 3$ for one of them ($Erasure(3)$). Independent of the disk size, the methods work under the assumption that $3/4$ of their disks will be filled with raw data. As the assumption is certainly incorrect for small disk spaces, any coding just wastes the space and results in a high overflow rate. Ideally, as we get into 500MB, nodes should be able to overcome the overflow problem completely. Results show that $Erasure(5)$ roughly does it. Comparing the miss ratio for any of the methods in 500MB and 900MB approximates the amount that the network misses due to overflow. This value is almost zero for $Erasure(5)$, while it is a substantial positive value for other methods.

E. Dependent Crashes

The assumption of independent crashes is acceptable in scenarios where the reason for the crash is localized. More serious causes of failure may result in damage of nodes in a specific area of the field, which contradicts independence of node crashes. Let "disaster" events occur in the field with a uniform distribution. Such an event ruins all nodes in a circle centered at the event and with a specific radius.

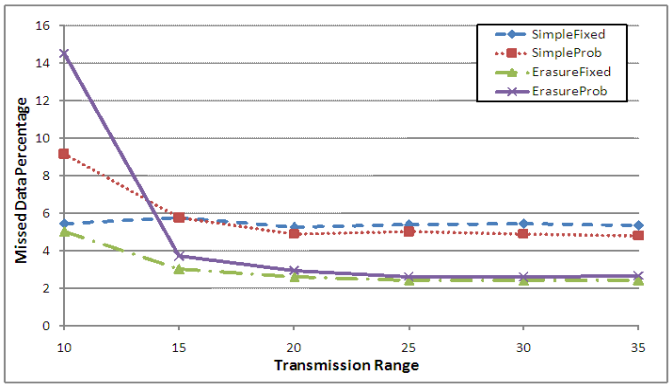

Fig. 10. Missed Data Fraction vs. Transmission Range

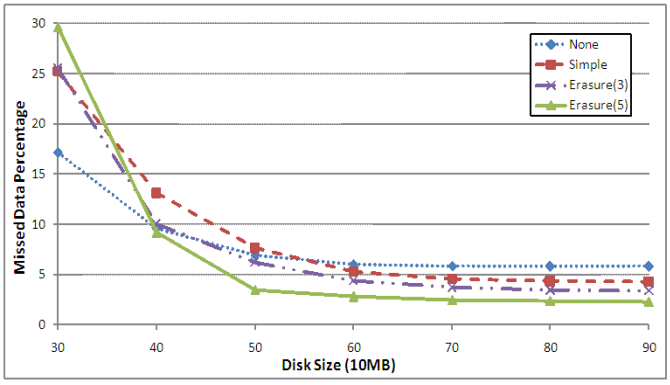

Fig. 12. Missed Data Fraction vs. Disk Size in fixed spreading

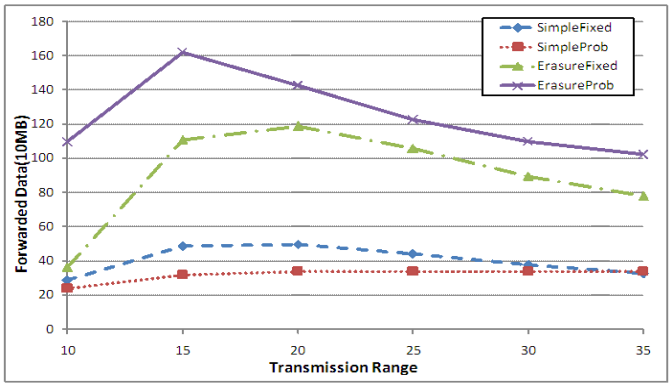

Fig. 11. Energy vs. Transmission Range

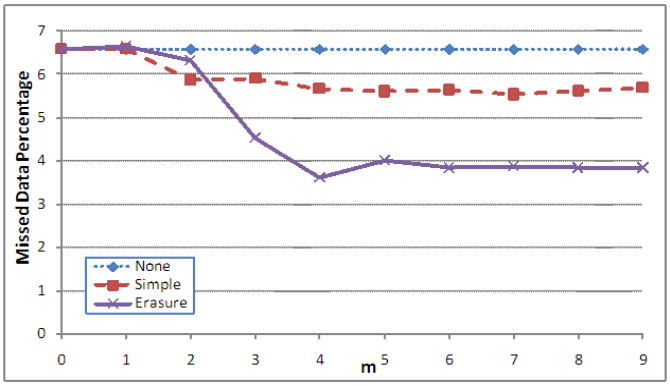

Fig. 13. Missed Data Fraction vs. m for dependent crashes

The expected value of this radius is set to 10 units in this experiment. The probability of the disaster is set so that the expected ratio of crashed nodes remains around 5 percent, similar to the above experiments. Figure 13 demonstrates the MDF vs. m. As formula (2) suggests, with growing of m, the independence assumption becomes more realistic. It can be seen by comparing figures 13 and 4. For $m \geq 5$ the independence approximation is fine. Clearly, for small m, one disaster may destroy all the regional data and there is no way to avoid it. Such events affect erasure coding more seriously (for small m). So, here simple coding works better for $m = 2$.

Overall, the evaluation demonstrate superiority of erasure coding over traditional replication. Erasure based methods help improve data retention by tolerating both crashes and overflows. During this section, we have also seen that the tunable property of our algorithm provides considerable flexibility to adjust the method for different environments. The probabilistic method shows acceptable performance for sufficiently dense underlying networks, while the more complex fixed method is better it for specific graphs. Also, we have presented a simulation showing that erasure based methods are better that simple redundancy for a substantial range of crash-probabilities. Finally, we considered environments with long-range disasters, where the crashes are not necessarily independent. It is shown that the performance of fixed erasure coding is only affected for very small values of m.

VI. Related Work

Sensor networks have been widely used for environmental monitoring applications. The larger volume of these applications are those dealing with low-bandwidth data (e.g. light [4], tempreature [5] and magnetic fields [6]). Sensor networks have also been deployed to sense high-bandwidth data. For example, in [7] sensors are deployed for structural monitoring in a building, and they have to record data at a frequency of 100Hz. Similarly, [8] deployed sensors close to a volcano, where they were sampling acoustic data at a hundred hertz.

In all the applications stated above, sensors transmit their sensed data continuously to a base station to be saved and processed if necessary. Consequently, the communication is the main challenge. In contrast, we consider in-network data storage during long periods of disconnected operation, which uses the memory of the sensors to keep as much data as possible considering the storage size and energy budget of the nodes. This approach proved helpful for delay tolerant networks such as [9] and [10].

Many previous articles investigated the storage problem on individual nodes; the file systems [11] and [12] are well-known examples. Distributed file systems have also been considered in past work (e.g. [13], [14]). The former applied a kind of distributed indexing of the data. More precisely, files are stored

on the nodes and the corresponding indices are created on the proxies. DIMENSIONS [15] applied the idea of storing data at different levels of resolution using compression techniques. In another work [14], all nodes send data to a base station in real time, while overhearing the data by some specific middle points contributes to system reliability. EnviroStore [16] is another effort that considered a partially connected network, where mules are occasionally connected to collect data. Ideally, the problem statement was how to distribute the data to maximize the expected amount collected by mules. This makes the problem similar to load balancing, which is another concern of our work. Load balancing has been used in many contexts in sensor networks. Balancing the energy consumption to maximize system life time [17], and balancing MAC layer access to improve fairness [18] are some examples.

Enviromic [1] is a recent project in sensor network, which addresses similar challenges. It is the most similar work to ours. Enviromic describes a sensor network deployed to collect raw acoustic data from the environment. With the main goal of minimizing the data loss, the authors designed a distributed file system. They took advantage of load balancing techniques to reduce the loss probability because of flash memory overflow. Their load balancing protocol is different, in that they had to consider notable resource constraints on sensor networks. In this paper, we add the dimension of reliability to the problem, were nodes may crash as another way of losing data.

There exists other work using different coding schemes to enhance reliability in sensor networks. Growth code, as a variant of LT codes [21] was suggested by [22] to maximize the total amount of recovered data at the base station. For data persistence, two recent results ([23] and [24]) applied fountain coding for large-scale sensor networks. The problem they considered is different from our problem in that they implemented distributed coding. Our problem does not require distributed coding since data to be coded originates at one node. As encoding is centralized in our application, we can pay in terms of rate and run time to get a code with small ϵ.

VII. Conclusion

We considered the reliability problem in storage-centric sensor networks in this paper. We distinguished two basic sub-problems; (i) how to code the raw data, and (ii) how to spread the blocks over the network. For coding, we tried simple and erasure redundancy. We demonstrated that the erasure coding scheme outperforms simple coding significantly. The other part of the problem is how to spread the coded block in the network to achieve a good trade-off between load balancing, energy conservation, and crash avoidance. We suggested two methods for this part. The first one is simpler as it works purely according to local information. The second one incurs more overhead but achieves better results, particularly for sparse networks. We analyzed our algoithms both mathematically and through simulation. The analysis and simulation show that they significantly improve the reliability of the system through better load balancing and fault tolerance.

References

[1] L. Luo, Q. Cao, C. Huang, T. Abdelzaher, J. A. Stankovic, and M. Ward, "Enviromic: Towards cooperative storage and retrieval in audio sensor networks," in *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2007, p. 34.

[2] M. Mitzenmacher, "Digital fountains: a survey and look forward," *Information Theory Workshop, 2004. IEEE*, pp. 271–276, 24-29 Oct. 2004.

[3] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[4] M. A. Batalin, M. Rahimi, Y. Yu, D. Liu, A. Kansal, G. S. Sukhatme, W. J. Kaiser, M. Hansen, G. J. Pottie, M. Srivastava, and D. Estrin, "Call and response: experiments in sampling the environment," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 25–38.

[5] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2002, pp. 88–97.

[6] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J. A. Stankovic, T. Abdelzaher, and B. H. Krogh, "Lightweight detection and classification for wireless sensor networks in realistic environments," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2005, pp. 205–217.

[7] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 13–24.

[8] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *USENIX'06: Proceedings of the 7th conference on USENIX Symposium on Operating Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 27–27.

[9] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi, "Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet," in *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*. New York, NY, USA: ACM, 2004, pp. 256–269.

[10] R. K. Ganti, P. Jayachandran, T. F. Abdelzaher, and J. A. Stankovic, "Satire: a software architecture for smart attire," in *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*. New York, NY, USA: ACM, 2006, pp. 110–123.

[11] H. Dai, M. Neufeld, and R. Han, "Elf: an efficient log-structured flash file system for micro sensor nodes," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 176–187.

[12] D. Gay, "Matchbox," 2003. [Online]. Available: http://www.tinyos.net/tinyos-1.x/doc/matchbox.pdf

[13] P. Desnoyers, D. Ganesan, and P. Shenoy, "Tsar: a two tier sensor storage architecture using interval skip graphs," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2005, pp. 39–50.

[14] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter, "Luster: wireless sensor network for environmental research," in *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2007, pp. 103–116.

[15] ——, "Luster: wireless sensor network for environmental research," in *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2007, pp. 103–116.

[16] L. Luo, C. Huang, T. Abdelzaher, and J. Stankovic, "Envirostore: A cooperative storage system for disconnected operation in sensor networks," *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 1802–1810, 6-12 May 2007.

[17] Q. Li, J. Aslam, and D. Rus, "Online power-aware routing in wireless ad-hoc networks," in *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2001, pp. 97–107.

[18] A. Woo and D. E. Culler, "A transmission control scheme for media access in sensor networks," in *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2001, pp. 221–235.

[19] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic load balancing on web-server systems," *IEEE Internet Computing*, vol. 3, no. 3, pp. 28–39, 1999.

[20] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica, "Load balancing in dynamic structured peer-to-peer systems," *Perform. Eval.*, vol. 63, no. 3, pp. 217–240, 2006.

[21] M. Luby, "Lt codes," in *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2002, p. 271.

[22] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: maximizing sensor network data persistence," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 255–266, 2006.

[23] Y. Lin, B. Liang, and B. Li, "Data persistence in large-scale sensor networks with decentralized fountain codes," *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 1658–1666, 6-12 May 2007.

[24] S. A. Aly, Z. Kong, and E. Soljanin, "Fountain codes based distributed storage algorithms for large-scale wireless sensor networks," *ipsn*, vol. 0, pp. 171–182, 2008.