Università di Pisa
Dipartimento di Informatica

Technical Report: TR-08-09

Replication vs Erasure Coding in Data Centric Storage for Sensor Networks

Michele Albano Stefano Chessa

May 21, 2008

ADDRESS: Largo B. Pontecorvo 3, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Replication vs Erasure Coding in Data Centric Storage for Sensor Networks

Michele Albano Stefano Chessa

May 21, 2008

Abstract

In-network storage of data in wireless sensor networks (such as DCS-GHT, for instance) is considered a viable alternative to external storage since it contributes to reduce the communications inside the network and to favor data aggregation. In current approaches it exploits pure data replication to assure data availability. In this paper, we consider the use of *n out of m* codes and data dispersal in combination to in-network storage. In particular, given an abstract model of in-network storage we show how *n out of m* codes can be employed, and we discuss how this can be achieved in three cases of study. Since the configuration of the *n out of m* and of the network is particularly critical with respect to correct data encoding, we define framework aimed at evaluating the probability of correct data encoding and decoding. Then we exploit this result and simulations to show how, in the cases of study, the parameters of the *n out of m* codes and the network should be configured in order to achieve correct data coding and decoding with high probability.

Keywords: Information Dispersal, Redundant Residue Number Systems

1 Introduction

In ubiquitous computing a network of disappearing devices, distributed at all scales unobtrusively provide assistance and services in the everyday people's life. In such a paradigm, wireless networks of sensors (WSN) [1] play an important role as they are the mean through which the surrounding environment can be monitored (thus providing feedbacks to the applications) and controlled. In a WSN a set of low-power, inexpensive embedded devices (called sensors) spontaneously cooperate to construct a network aimed

at monitoring and control tasks. Each sensors, which is a microsystem combined with a radio interface, can measure environmental parameters by means of on-board transducers or can control the environment by means of actuators. A special sensor, called *sink node*, acts as a gateway with the external network to the purpose of data collection and network control and programming.

There are a number of challenges arising in the design of efficient and scalable WSN, however an important issue that emerges from these challenges is energy consumption, and its dual, networks' lifetime [11]. In fact sensors rely on on-board batteries for energy supply, and, once a sensor depletes its battery, it becomes unreachable from the other sensors and may be considered faulty at any effect [12].

The most established model for WSNs, Directed Diffusion [8], employs a query distribution and data collection algorithm implementing an external storage scheme, where sensed data is sent to the sink node for storage. This model assumes that the sink node has a permanent connection with the network, and it performs most of the data analysis, while the role of the WSN is limited to data acquisition and, in some cases, to simple data processing. This assumption is motivated by the fact that, with the current technologies [9], sensors are unable to perform complex data processing and storage. This model essentially consider the WSN as an extension of the sink node, which coordinates the activity of the network and issues queries to the sensors. However this hypothesis limits the applications of WSNs. In fact in some applications the connection between the sink node and the WSN may be unavailable for (arbitratily long) periods, while in other cases it is not practical neither efficient to accumulate all the data in the sink node. For these reasons [5] introduced the DCS-GHT model, in which data is stored within the WSN. Comparing this approach to the external storage approach, the author observed that in-network storage may contribute to save sensors' energy and to improve the network lifetime. Furthermore in-network storage naturally permits the coexistence of multiple sink nodes.

Since sensors have limited memory capacity, the storage of all the data sensed by the WSN may result impractical. However, many WSN applications produce datasets of limited size that are the results of in-network data processing and aggregation strategies on a large amount of "raw" sensed data.

To the best of our knowledge, all current approaches to in-network storage adopt pure replication [5, 6, 7, 8, 17]. On the other hand, it is a well-known fact that in many applications pure replication leads to a significantly larger memory overhead as compared to solutions based on erasure

codes [15, 16]. Erasure code is a general name for the techniques that split the data into (redundant) parts and guarantee the survival of data in front of the erasure (loss) of a number of its parts, for this reason erasure codes are also known as *n out of m* codes. They are based on a mathematical technique that, given a data and m keys, computes m fragments (one for each key), with the property that the original data can be reconstructed from any subset of n fragments provided the keys used to construct each of these fragments is known.

In this paper we we reconsider the in-network storage approach to show how it can employ erasure codes. In particular we propose an abstract model of in-network storage and we show how it can integrate an erasure encoding, then we describe how this can be achieved in two specific in-network protocols. To this purpose we propose a mechanism for data dispersal that exploit erasure codes, and we define a probabilistic model to evaluate the probability of a correct coding and decoding of the fragments in front of failing sensors, depending on the parameters n and m of the erasure code, on the WSN configuration, and on the way in which the erasure code keys are distributed to the sensors. Using the model, we show that said probabilities are high even for key distribution setting that are feasible in realistic scenarios. In the end, we use analytical results and simulations to compare the performance of two protocols for network storage when they use replication or erasure code, and to show that the erasure codes give benefits in terms of memory overhead, but their use requires a slightly larger communication overhead.

The rest of this paper is organized as follows. Section 2 presents previous results about both in-network data storage and data availability for WSN, Section 3 describes in detail an erasure code based on Redundant Residue Number Systems (*RRNS*). Section 4 discusses the use of general *n out of m code* in WSNs, and Section 5 depicts concrete architectures based on local storage and DCS that use RRNS. Section 6 performs the analysis of the general model, Section 7 applies the analysis to the concrete architectures of Section 5. Finally, Section 8 discuss the memory overhead of systems based on erasure codes, and the conclusions are drawn in Section 9.

2 Related Works

2.1 Data centric storage

In many applications sensed data is generally more important than the sensors that have sensed it. Based on this observation the *Data Centric Storage*

(DCS) [5], defines a paradigm where the data is stored within the network itself, in a set of sensors that is selected based on the meta-datum that describes the data. In particular each data is associated to a meta-datum and the data is stored in a set of sensors that is a function of the meta-data.

There are a number of different proposals of DCS, that differ for the way in which:

1. the data is assigned a meta-datum;
2. the sensors that store the data described by a meta-datum are selected
3. the data is routed to/from the sensors that store it.

The reference model of DCS is the *Geographic Hash Table* (GHT) [5], that constitute the first proposal of DCS. In the DCS-GHT it is assumed that the geographic coordinate of each sensor is known, and that each data is described by a unique *meta-datum* (or *name*). The set of sensors selected to store a data is computed by means of an hash function applied to the corresponding meta-datum that returns a pair of geographic coordinates fitting in the area where the sensor network is deployed.

DCS-GHT implements two primitives: `put`, which stores data, and `get`, which retrieves it. In the `put` primitive, the meta-datum of a data is hashed to a pair of coordinates (x, y). Then, DCS-GHT routes a packet containing the data and its meta-data to the (x, y) coordinate by means of the GPSR routing protocol [3]. In general the (x, y) does not correspond to any sensor, however GPSR can route the packet to the sensor (denoted *home node*) closest to (x, y), and it also identifies a perimeter of sensors (denoted *home perimeter*) surrounding the point (x, y) that also include the home node. Then DCS-GHT stores a copy of the data in all the sensors in the the home perimeter, to guarantee data persistence in presence of sensor faults.

Data retrieval is performed by means of the `get` primitive. This primitive takes in input a meta-datum k, computes the corresponding coordinate $(x.y)$ by means of the same hash function used by the `put`, and uses GPSR to send a request for any data of meta-datum k. When the request reaches any sensor in the perimeter surrounding (x, y), this sensor responds by sending back to the requested all data of meta-datum k it stores.

Along this trend of research many alternative DCS mechanisms, such as Cell Hash Routing (*CHR*) [23], Graph EMbedding (*GEM*) [24], Hierarchical Location Routing (*LHR*) [25], and Q-NiGHT [7, 6], have been proposed. They are similar to DCS-GHT in the definition of the `put` and `get` primitives, and they differ in the internal mechanisms used to implement routing, data dispersal and storage.

In particular Q-NiGHT addresses also non-uniformly distributed WSN and introduces the concept of QoS in the data availability. To this purpose Q-NiGHT defines a different dispersal protocol that stores the replicas of the data in a ball of sensors centered in the home node (i.e. a set of sensors within a given distance from the home node) rather than in the home perimeter. In this way Q-NiGHT can control (and thus limit) the number of replicas of the data. The authors show that in this way the memory overhead and load on sensors is more balanced than DCS-GHT.

2.2 Data Availability

All the previously cited approaches to the storage in WSNs adopt pure replication, that is the replication of the whole data on every sensor of the set selected for the storage.

On the other hand, beginning with the seminal work of Shannon [26] in 1948, a number of efficient redundancy techniques, and in particular erasure codes have been employed in many application areas [27].

The Information Dispersal Algorithm (IDA) [30] is a well known example of erasure code. In IDA a data d of size l symbols is encoded into a set of $m = n + r$ fragments $s_1, \cdots, s_m$,with the property that d can be reconstructed form any subset of n fragments. Furthermore d can be reconstructed if up to $e < r$ fragments are lost and up to $c = \lfloor \frac{r-e}{2} \rfloor$ fragments are corrupted. IDA is optimal with respect to code efficiency since the size of each fragment is almost equal to the size of a symbol. Furthermore it allows for efficient encoding and decoding procedures. The complexity of encoding is $O(l)$, while the complexity of decoding is $O(l(\log n + r))$. A similar result holds for erasure codes based on Redundant Residue Number Systems (RRNS) [29]. Turbo codes [28] are other examples of erasure codes. As compared to IDA they require linear time encoding and decoding procedures, but they are less space efficient.

Erasure codes have also been proposed for application in WSN [21], where it is proposed a new erasure code based on linear coding. The authors show that the set-up communication cost scales as the logarithm of the number of sensors that store data.

3 Redundant Residue Number Systems

Redundant Residue Number Systems (*RRNS*) can be used to implement an erasure code as follows. Given $m = n + r$ pairwise prime, positive integers m_1, ... , m_{n+r} called moduli, let $M = \Pi_{p=1}^{n} m_p$, $M_R = \Pi_{p=n+1}^{n+r} m_p$, and,

without loss of generality, $m_p > m_{p-1}$ for each $p \in [2, n]$. Given any non-negative integer X, let $x_p = X \bmod m_p$ be the residue of X modulo m_p.

Hereafter, $(a)_b$ denotes a mod b. Furthermore, given three integers a,b,c, it is said that a is congruent to b modulo c (denoted $a \equiv b \bmod c$) if $a \bmod c = b \bmod c$.

The number system representing integers in $[0, M)$ with the $(n+r)$-tuples of their residues modulo m_1, ... ,m_{n+r} is called the Redundant Residue Number System (*RRNS*) of moduli m_1, ... ,m_{n+r}, range M and redundancy M_R [15, 16]. For every $(n+r)$-tuple (x_1, .. ,x_{n+r}), the corresponding integer X can be reconstructed by means of the Chinese Remainder Theorem [22].

$$X = \left(\sum_{p=1}^{n+r} \left(\frac{MM_R}{m_p} \left(x_p \beta_p\right)_{m_p} \right) \right)_{MM_R}$$

where, for each $p \in [1, n]$, $\beta_p = \left\langle \frac{MM_R}{m_p} \right\rangle_{m_p}$ is the multiplicative inverse of MM_R/m_p modulo m_p, that is, $\left(\frac{MM_R}{m_p} \right)_{m_p} = 1$, and β_p is in the range $[0, m_p)$.

Although the given RRNS can provide unique representations to all integers in the range $[0, MM_R)$ [15, 16], the legitimate range of representation is limited to $[0, M)$, and the corresponding $(n + r)$-tuples, are called legitimate. Integers in $[M, MM_R)$ and the corresponding $(n+r)$-tuples are called illegitimate.

Given an RRNS of range M and redundancy M_R, with moduli m_1, ... ,m_{n+r}, let $(x_1, ..., x_{n+r})$ be the legitimate representation of some X in $[0, M)$. An *erasure* of multiplicity e is an event making unavailable e arbitrary digits in the representation, and an *error* of multiplicity c is an event transforming c arbitrary, unknown digits. If $e + 2c \leq r$ then the RRNS can correct the errors to reconstruct X [16].

If the moduli are single precision integers, the coding/encoding operations can be executed in linear time using single precision operations, and the code efficiency in terms of storage overhead is optimal or nearly optimal in practical applications. This means that, if the range of representation of the data is $[0, M)$, thus each data can be represented by $L = \lceil \log_2 M \rceil$ bits, and the largest modulo of the RRNS used for encoding is m_P, then each fragment can be represented by at most $L_n = \lceil \log_2 m_P \rceil$, where $L_n \sim L/n$ [17].

4 An in-network storage model

In-network storage approaches associate to each data instance d a meta-data k (denoted by d:k), and offer to the sensors' applications the two primitives `put(`d`:`k`)` (to store d:k) and `get(`k`)` (to retrieves any data whose meta-data matches k).

The `put(`d`:`k`)` primitive first selects the set of sensors N_k as a function of k; then it multicasts a storage request of d:k to the sensors in N_k. In turn, each sensor in N_k stores an encoding of d:k, denoted $f(d{:}k)$, in its internal storage.

Data retrieval is performed by means of the `get` primitive. Given a meta-data k, it first computes the set of sensors N_k that store the corresponding data. Then it sends a request to the sensors in N_k that contain the meta-data k. The sensors in N_k reply to this request by sending all data of meta-data k to the sensor performing the `get`.

A generic sensing task of a sensor that cyclically senses data with meta-data k is thus represented by Algorithm 1. When a data d:k is sent to N_k for storage, upon reception of the corresponding storage request message, a generic sensor in N_k executes Algorithm 2.

Once a sensor in N_k receives a request for data of meta-data k, it extracts from its memory all the matching data and sends these data to the requester, as shown in Algorithm 3.

Table 1: Sensing task of sensor p:

loop ...
sense data d of meta-data k;
put(d:k);
end loop

Table 2: On reception of d:k:

$x = f$(d:k)
store x:k in its memory

In Algorithm 2, each sensor p in N_k applies an encoding function f to the the data, then it stores encoded data in its memory. In most of the current systems, function f is the identity function, hence the sensors store $f(d) = d$ in their memory. However, we observe that function f can implement an

Table 3: On reception of a request from sensor p for meta-data k:

```
foreach stored pair d:k'
  if match(k',k)
     send d:k' to p
```

arbitrary *n out of m* erasure code. In this case, each sensor would store a fragment of the original data. Due to the property of the erasure codes, each fragment occupies a memory space smaller than the space required to store the original data.

The implementation of a function f based on an *n out of m* code also requires that each sensor be assigned with a key used for the encoding (in the case of the RRNS this key is a module). Recall that the association between a fragment and the key used for the encoding should be kept to ensure a correct data reconstruction; recall also that correct reconstruction is guaranteed if at least n fragments produced with different keys.

When a data d:k stored by sensors in N_k is requested by a sensor, the reconstruction of the data can be performed according according to the following strategies:

1. each sensors in N_k retrieves in its memory the fragment x:k corresponding to d:k and sends it to the sink node. In turn the sink node receives all the fragments corresponding to d:k and reconstructs the data.

2. or one sensor in N_k collects all the fragments corresponding to d:k and stored by sensors in N_k. This sensor reconstructs d:k and it sends the reconstructed data d:k to the sink node.

These two strategies have different advantages and drawbacks:

- Strategy 1: there is a greater energy usage because each sensor storing a fragment sends the fragment to the sink node, on the other hand the transfer of these fragment is more reliable: even if some of the fragments are lost, as far as n of the m different fragments are received by the sink node, it is possible to reconstruct the data. The encoding also provides basic encryption of the encoded data, in fact, once data is encoded, it is inaccessible from external entities eavesdropping the wireless channel during the communication between the sensors and the sink node.

- Strategy 2: it requires only local communications to reconstruct the data and only one packet with the data is sent to the sink node. However the transfer of the data to the sink node in this case is less reliable.

5 Data Dispersal & Retrieval based on Erasure Codes

In this Section we consider erasure codes based on RRNS and we show how they can be integrated in some known in-network storage protocols. In particular we consider the case in which the storage is in an area local to the node that produce the data (called local storage), and two Data Centric Storage systems, namely DCS-GHT [5] and Q-NiGHT [6].

Since RRNS are used for encoding, each sensor is assigned with a module to be used to compute the fragments (that in this case are residues). We assume that for each sensor the assigned module is randomly chosen (possibly at application compile time) from a library of $m = n + r$ pairwise prime moduli $m_1, ..., m_{n+r}$, where $m_i > m_{i-1}$ for each $i \in [2, n + r]$. Hereafter notation $m(p)$ denotes the modulo assigned to sensor p.

Once a sensor p assigned with modulo $m(p)$ receives for storage the pair $d{:}k$ (thus p belongs to N_k), it stores in its memory the pair $x{:}k$ where x is the fragment given by the residue of d module $m(p)$.

When another sensor (say, the sink node) sends a request for any data matching the meta-data k to the sensors in N_k, the sensor p retrieves $x{:}k$ (that matches k) and sends $x{:}k$ to the requester.

5.1 Local storage

Local storage is a very simple storage protocol in which the data is stored into all the sensors that are reachable by the producer in one hop. That is, when a sensor p produces a data $d{:}k$, it sends the data in local broadcast to all the sensors that are reachable in one hop, thus in this case N_k is the set containing p and all its one-hop neighbors.: Each sensor $p_j \in N_k$ thus computes the residue $x_j = d{:}k \mod m(p_j)$ and stores it in its memory.

Clearly in this case any request for a data should be directed directly to the node that produced it (under this respect this approach is node-centric). In terms of communications this is less efficient then DCS, however we keep this simple model with the purpose of comparison and evaluation of memory overhead.

5.2 Data Centric Storage

The main difference between local storage and Data Centric Storage is that the set N_k of the sensors that are selected to store the data $d{:}k$ is obtained by the application of an hash function to the meta-datum k. The hash function returns a geographical point in the sensing field, i.e.: the *home node*, and the sensors of set N_k are either:

- for DCS-GHT, the sensors belonging to the home perimeter around the home node;
- for Q-NiGHT, the q_k sensors closest to the *home node*. q_k is an indication of the QoS requested for the data of meta-data k in the storage, as it reflects the number of sensors that store the corresponding data.

In both cases each node in N_k stores a fragment of the data computed with an *n out of m* RRNS code. Thus the `get` primitive is implemented by a request message that should reach at least n sensors with different moduli.

Then these sensors provide the data to the requester by using either strategy 1 (i.e. they all send their fragments to the requester, which in turn reconstructs the data) or strategy 2 (i.e. they reconstruct the data with a distributed algorithm and send the data to the requester).

5.3 Encoding example

Here is presented an example of encoding using RRNS.

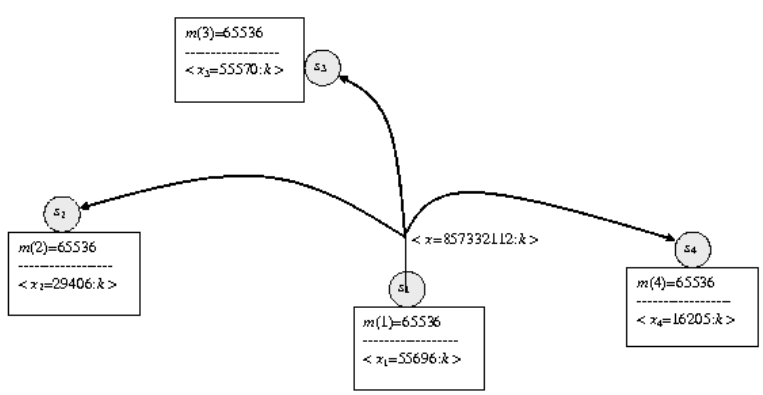

Figure 1: Residue Storage in a WSN

Let us consider an RRNS with $n = 2$ and $r = 2$ given by moduli $m_1 = 65536$, $m_2 = 65533$, $m_3 = 65531$, and $m_4 = 65529$. Moduli m_1 and m_2 are the redundant moduli, and the range of the RRNS is $M = 4294180899$.

Let us consider a sensor p_1 with 3 neighbors p_2, p_3, p_4 assigned with moduli m_1, m_2, m_3, m_4, respectively. When sensor p_1 produces a data $d = 857332112$, it broadcasts $d{:}k$ to all of its neighbors, it computes $x_1 = d \bmod m_1 = 55696$, and it stores in its memory $x_1{:}k$.

In turn, when a neighbor p_j receives $d{:}k$ from sensor p_1, it computes $x_j = d \bmod m_j$ and stores $x_j{:}k$ in its memory. The values of x_j are $x_2 = 29406$, $x_3 = 55570$, $x_4 = 16205$, respectively. This situation is shown in Figure 1.

If p_1 and p_3 fail, when the sink node connects to the network the memory of sensors p_1 and p_3 are no longer available, hence the sink node have access only to $x_2 = 29406$ and $x_4 = 16205$ from x_2 and x_4, respectively. However, since the RRNS can afford up to r erasures, the original value of d can still be recovered by the sink node applying the Chinese Reminder Theorem to residues $x_2 = 29406$ and $x_4 = 16205$ with moduli $m_2 = 65533$ and $m_4 = 65529$, respectively.

6 Probabilistic Model

In this section we evaluate the probability of correct retrieval of a data $d{:}k$ from the corresponding fragments stored in the sensors in N_k.

We recall that in a *n out of m* coding, it is possible to decode d only if at least n residues with different moduli are available. To this purpose it is then necessary to ensure the following property:

Property 1 *For each meta-datum k there exist at least n sensors $p_1, ..., p_n \in N_k$ such that $m(p_u) \neq m(p_v)$ for each $u, v \in [1, n]$, $u \neq v$.*

In practice, since sensors may fail, the fragments they store might become inaccessible, and this may impair property 1, thus preventing the sink node from decoding $d{:}k$. To this purpose, we require that the following property is satisfied:

Property 2 *For a given meta-datum k, there exist at least l (with $l \geq n$) sensors p_1, ... , $p_l \in N_k$ such that $m(p_u) \neq m(p_v)$ for each $u, v \in [1, l]$, $u \neq v$.*

Property 2 introduces a new parameter l (called local redundancy) which determines the amount of different moduli used to encode the data $d{:}k$ in

the set N_k. Property 2 ensures that, as long as no more than $l - n$ sensors of N_k fail, then the data $d{:}k$ can be correctly recovered by the sink node.

In a WSN, Property 2 can be ensured deterministically (provided that at least l sensors are used to store data of each meta-datum, that is, each N_k has at least l elements) during the network deployment. In particular the network could execute a distributed algorithm to initialize the moduli of the sensors such that Property 2 is satisfied. This however would introduce additional communication and computational overhead on the sensors.

For this reason we consider a probabilistic approach, in which the moduli are assigned to the sensors during the manufacturing process, and the sensors are randomly scattered throughout the sensing field. Depending on the parameters of the network, namely the number of sensors in the network, the number m of moduli in the library, the local redundancy l, the transmission range t, and the side L of the sensing field, we evaluate the probability that, for a given meta-data, Property 2 holds. Acting on these parameters, the network manager can achieve the desired level of fault tolerance.

In the next section we exploit this result to evaluate the probability of correct data reconstruction in three different storage strategies (local storage, DCS-GHT, or Q-NiGHT).

Given the set of N_k sensors and a library of m modules, we first evaluate the probability that the sensors in N_k have exactly l different moduli. Let n_k be the cardinality of N_k (i.e. $n_k = |N_k|$), clearly there are m^{n_k} different ways of selecting n_k moduli randomly chosen in a library of m moduli. Let $\lambda(m, l)$ be the number of sequences of n_k moduli chosen from the library of m moduli that contain exactly l different moduli. $\lambda(m, l)$ can be evaluated by considering the number of ways it is possible to select l different moduli among a library of m moduli, i.e. $\binom{l}{m}$, multiplied by the number of ways that l moduli can be disposed: $l^{n_k} - \sum_{j=1}^{l-1} \lambda(l, j)$. From which:

$$\lambda(m, l) = \binom{l}{m} \left(l^{n_k} - \sum_{j=1}^{l-1} \lambda(l, j) \right)$$

It is immediate that $\lambda(l, 1) = l$ (which corresponds to the l cases where all the n_k sensors are assigned with the same modulo). From this analysis follows that the probability that Property 2 holds, i.e. the probability $\gamma(l, n_k)$ that in N_k there are at least l different moduli can be expressed as:

$$\gamma(l, n_k) = \frac{\lambda(m, l)}{m^{n_k}}$$

Recall that if Property 2 holds for redundancy $l \geq n$, then the data produced by the sensor can be recovered by the sink node provided the number of faults in the neighborhood N_k does not exceed $l - n$.

In order to evaluate the probability of correct encoding and decoding, let $\phi(n_k)$ be the density of probability of n_k, and let $\theta(n_k)$ be probability that there are at least n different moduli in a given set N_k of cardinality n_k. We have that:

$$\theta(n_k) = 1 - \sum_{i=0}^{l-1} \gamma(i, n_k)$$

Thus the probability θ of correct data encoding (i.e. the probability that in the set chosen for storage that are at least n different moduli) can be expressed as:

$$\theta = \sum_{i=n_k}^{Z} (\phi(n_k)\theta(n_k))$$

7 Cases of study

The probabilistic analysis provided in the previous section can be applied to different WSN scenarios. In this section we compose the previous results to evaluate the probability to correctly reconstructing a data encoded with an RRNS and stored according to the Local storage, DCS-GHT, and Q-NiGHT protocols.

7.1 Local storage

The local storage protocol dictates that a data of meta-data k is stored in the neighborhood N_k of a node p, i.e. in the nodes within a single hop from p.

Let Z be the number of sensors and let us assume that the sensors are deployed in a square sensing field of side L. Let also n_k be the size of N_k, and t be the transmission range of each sensor. The probability that a sensor has exactly n_k neighbors is:

$$\phi(n_k) = \binom{Z-1}{n_k-1} \left(\frac{\pi t^2}{L^2}\right)^{n_k-1} \left(1 - \frac{\pi t^2}{l^2}\right)^{Z-n_k}$$

In order to evaluate $\phi(n_k)$ we assume that the probability that a sensor q is a neighbor of a given sensor p is $\pi t^2/l^2$. The underlying hypothesis

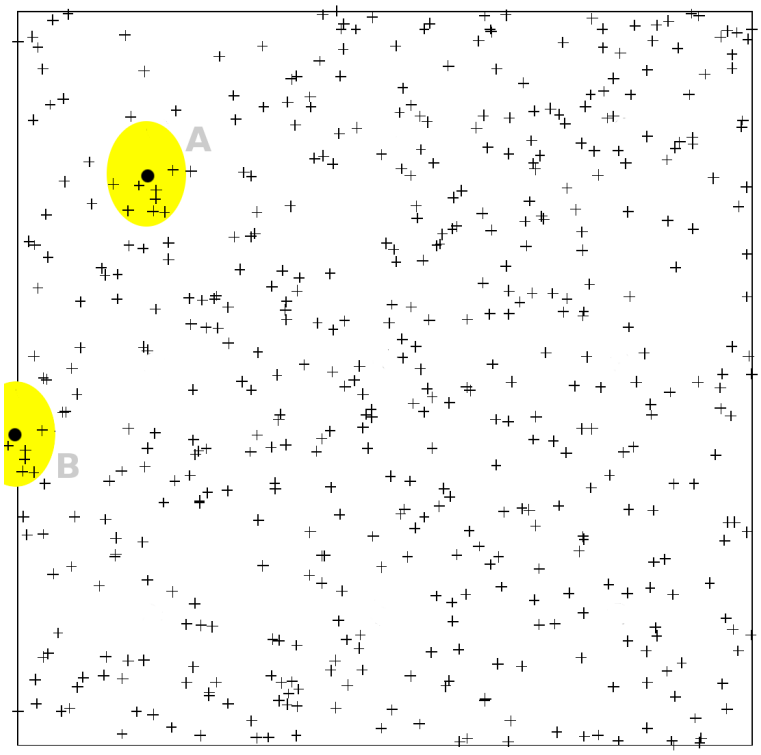

Figure 2: Residue Storage in a WSN

is that the intersection between the sensing field and the area covered by the transmission range of a sensor is always circular with radius t. The formula neglects border effects. In fact, as shown in the Figure 2, a sensor in the network border (as it is the case B has, in general, a smaller number of neighbors. On the other hand the approximation resulting from this assumption does not impair the asymptotical result.

Figure 3 depicts the distribution of the redundancy l for a WSN characterized by a sensor density of 30, using *5 out of 15* codes. The figure shows that l is greater than 10 with high probability, and the resulting WSN can cope with 5 erasures.

7.2 DCS-GHT

The evaluation of the probability of a correct reconstruction of the encoded data is more difficult for DCS-GHT. In this scenery, the number of sensors

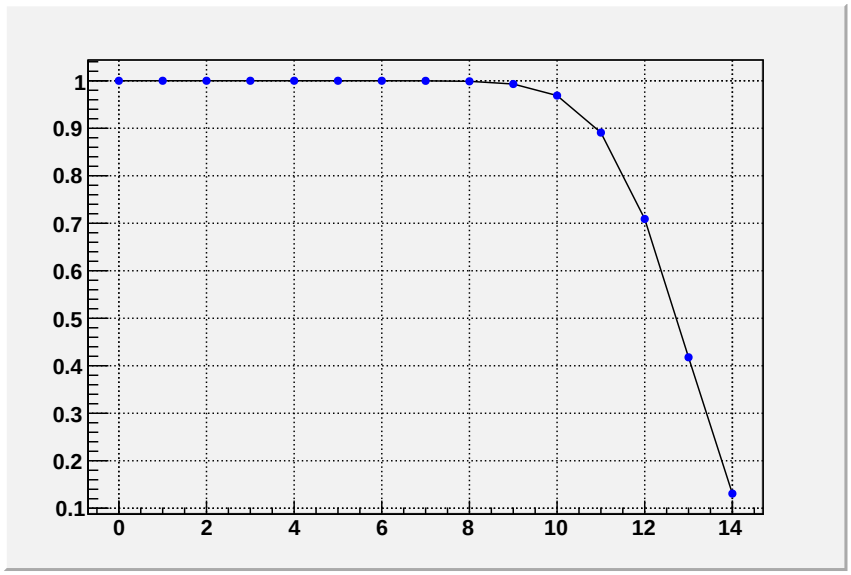

Figure 3: Probability θ for 5 out of 15 codes, sensor density = 10, and $l \in [5, 15]$

in N_k is the number of sensors that belong to the perimeter around an *home node*. In general, it is not possible to estimate it with an analytical formula. For this reason we evaluate the size of $N-k$ by means of simulation. In particular we perform simulations aimed at producing a table with the distribution of the number of sensors on a perimeter, and we combine the data in this table with the formula of the previous Section. This approach is phenomenological and it is based on monte carlo simulations.

We performed simulations with the following settings:

- the WSN is deployed in a square sensing field with length $L = 400$ m
- sensors have a trasmission radius $t = 10$ m
- sensors' density is defined as the mean number of neighbors for each sensor, that is $\rho = n\frac{\pi t^2}{L^2}$
- sensors are uniformly distributed
- density is in the range $[7, 40]$
- for each density, 100 different WSNs were generated

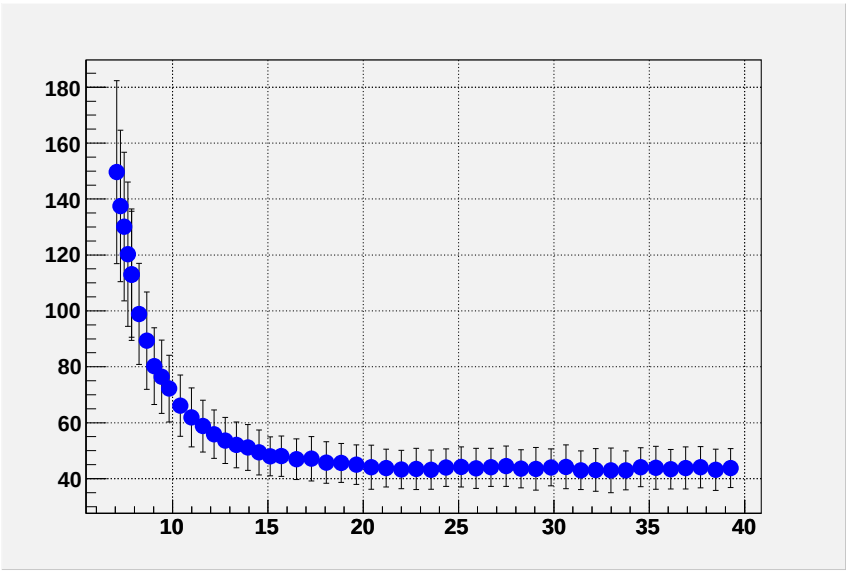

Figure 4: Mean perimeter

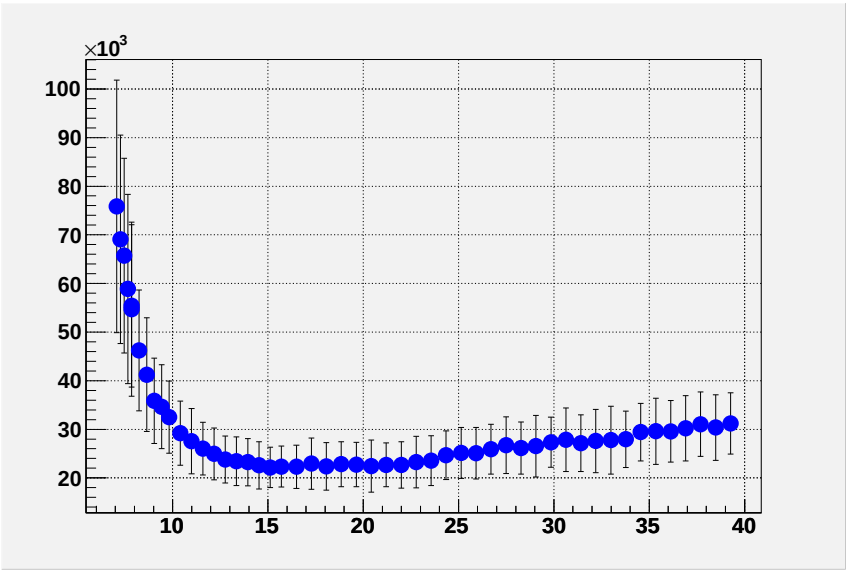

Figure 5: Mean variance

- for each WSN, 1000 `put` operations were simulated

The results of these simulations are shown in figures 4 and 5 that report the mean perimeter size and its variance, respectively.

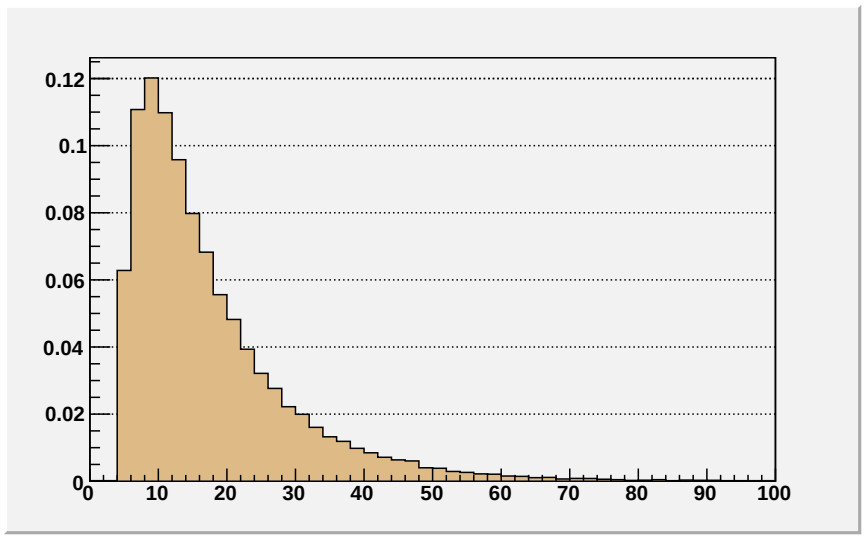

Figure 6: Distribution of n_k for sensor density = 10

Figure 6 shows the distribution $\phi(n_k)$ of the number of nodes in N_k for a given meta-datum k in a WSN with sensor density 30. These data and the shown values can be put into a table, that is the distribution $\phi(n_k)$. Given the distribution $\phi(n_k)$, the probability that in the storage set of a given meta-datum k with n_k sensors there are at least l different moduli (with $m \geq l \geq n$), like for the local storage, is:

$$\theta = \sum_{i=n_k}^{Z} (\phi(n_k)\theta(n_k))$$

Figure 7 depicts the distribution of the redundancy l for a WSN characterized by a sensor density of 30, using *5 out of 15* codes. The figure shows that l is greater than 7 with high probability, and the resulting WSN can cope with 2 erasures.

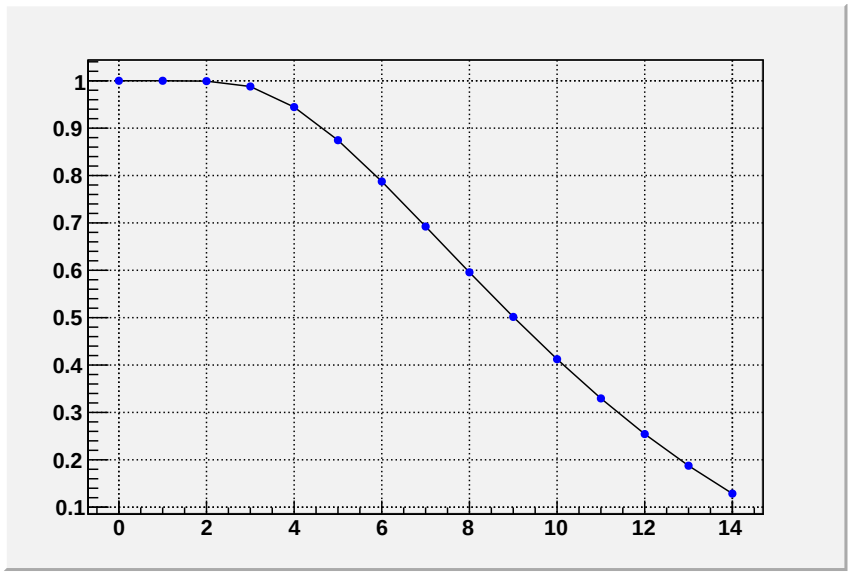

Figure 7: Probability θ for 5 out of 15 codes, sensor density = 10, and $l \in [5, 15]$

7.3 Q-NiGHT

In the case of Q-NiGHT, the initial number of sensors in N_k is q_k, that is the number of replicas that was associated with the meta-datum k during the dissemination of the data. It is then possible to describe the distribution of the number of sensors with a Kronecker delta

$$\phi(n_k) = \begin{cases} 1 & \text{when } n_k = q_k \\ 0 & \text{elsewhere} \end{cases}$$

that is a function that is always 0, except when its argument is equal to q_k, that results in a 1.

Combining the formulas is simple, in fact the probability that in the neighborhood of a given meta-datum k with n_k neighbors there are at least l different moduli (with $m \geq l \geq n$) is:

$$\theta = \sum_{i=n_k}^{Z} (\phi(n_k)\theta(n_k)) = \theta(q_k)$$

Figure 8 depicts probability θ that property 2 holds for a given meta-

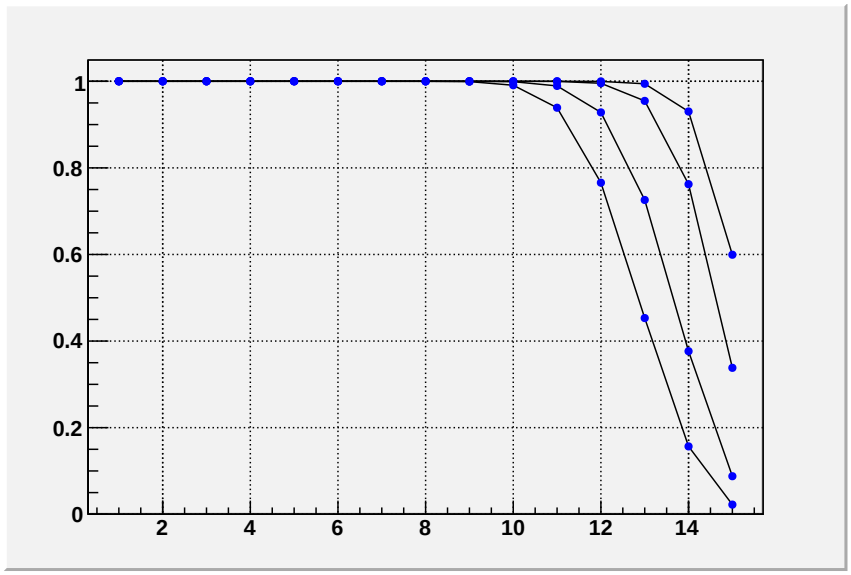

Figure 8: Probability θ for 5 out of 15 codes, $n_k \in 25, 30, 40, 50$, and $l \in [5, 15]$

datum k, using *5 out of 15* codes, for different values $l \in [5, 15]$ and $n_k \in \{25, 30, 40, 50\}$. It is seen that as l increases the probability θ decreases, but it remains close to 1 as long as $l \leq 10$ for $n_k = 25$ and $l \leq 13$ for $n_k = 40$. This means that, when $n_k = 40$, the probability of recovering the information of a given meta-datum l is close to 1 if at most 8 out of n_k sensors fail.

Furthermore, as the number of sensors in N_k increases, this contributes to increase probability θ.

A similar behavior can be observed in Figure 9 which depicts probability θ evaluated for 10 out of 20 codes, $l \in [10, 20]$ and $n_k \in \{25, 30, 40, 50\}$.

8 Memory overhead

This Section presents a comparison of pure replication and *n out of m* coding with respect to the memory savings implemented by the coding strategy.

Let us assume that the data to be stored are represented by L bit, that the RRNS used for the erasure encoding is an *n out of m* code, thus each data is encoded into fragments of size $L_n \sim L/n$ bits, and that the data is

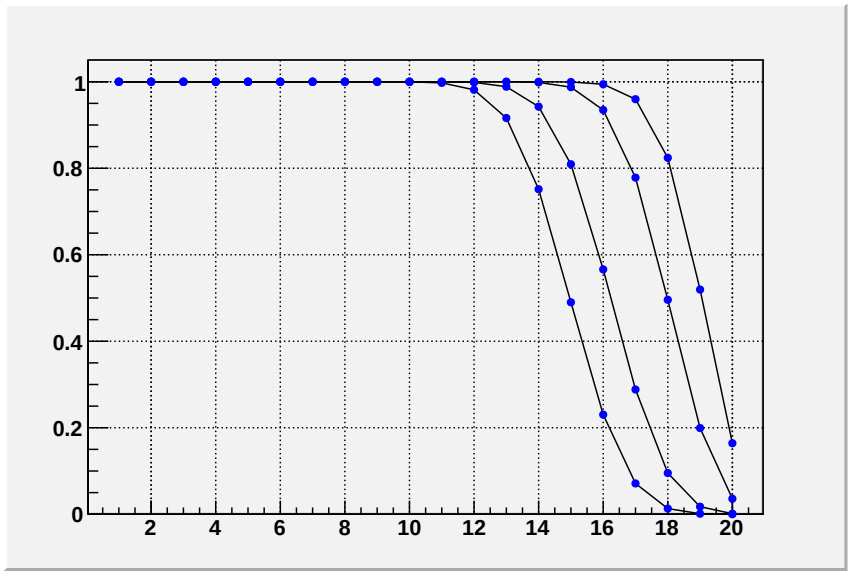

Figure 9: Probability θ for 10 out of 20 codes, $n_k \in 25, 30, 40, 50$, and $l \in [10, 20]$

stored in a set N_k of n_k sensors (selected by a local or a DCS storage).

It is immediate that with replication the memory overhead per data per sensor is L bits, while with the RRNS is $L_n \sim L/n$ bits. The total memory overhead is Ln_k for the replication strategy, $L_n n_k = L\frac{n_k}{n}$ for the n *out of* m coding, hence the erasure codes strategy outperforms the pure replication one for a factor n on the memory usage. On the other hand, the pure replication strategy is exceedingly redundant. Since both the DCS-GHT and the local storage cannot choose n_k, it is just a matter of having network parameters that enable the correct reconstruction of the data w.h.p., and under this condition the erasure coding outperforms the replication strategy.

The case of Q-NiGHT is more tricky. In fact, it assigns to a given metadatum a value of n_k that grants an high enough probability of reconstructing the corresponding data. Hence, if a data should survive up to r erasures, pure replication strategy will select a n_k of $r + 1$, while the RRNS strategy will resort to the results of previous section, looking for parameters that can perform the right level of redundancy. Then it is sufficient to add r to the selected level of redundancy to determine the number of sensors that must store the data. Should r sensors fail, N_k will have nonetheless an high enough level of redundancy to be sufficient to reconstruct the data.

For example, let us suppose that the goal of the storage system is to survive up to 7 erasures with high probability. From the analysis of Section 7, it is clear that *n out of m* codes with $n = 5$ and $m = 15$ can cope with this fault level in a WSNs using Q-NiGHT with QoS parameter = 30. On the other hand, pure replication can be a feasible alternative in WSNs using QoS parameter = 8.

On the memory usage side, if the data to be stored is initially encoded using L bits, pure replication solutions will use $8L$ bits, with L bits used on 8 different sensors. Solutions using *n out of m* codes will employ $L_n = L/5$ bits on each of the 30 sensors, for a grand total of $6L$ bits used.

Hence the use of *n out of m* codes presents advantages in terms of global usage of memory, and provides a better QoS, i.e. each sensor has to store a lesser number of bits, resulting in a better exploitation of the inherent redundancy of the network, especially when the data are related with a low number of meta-data: pure replication would store all the data in a low number of sensors, while *n out of m* codes would use a larger number of sensors, with a lesser occupancy on each sensor.

9 Conclusions

The Data Centric Storage systems are very effective in implementing an in-network data storage and retrieval system, since they require only unicast communications. In this paper we have shown how Data Centric Storage systems can be combined with memory-efficient erasure codes. The use of erasure codes is however not immediate, since it requires the sensors to perform the encoding of the data before the storage (encoding that is not necessary in traditional DCS since they exploit pure replication). In fact it is necessary that each sensor be assigned with a coding parameter (that in the case of RRNS is a module), and this assignment is critical from the point of view of correct data coding and decoding. For this reason we have proposed a probabilistic model that allows the estimation of correct coding and decoding probability, and we have shown how this model can be adapted to estimate this probability in three cases of study: local storage, DCS-GHT, and Q-NiGHT. From the analytical and simulative results we showed how correct coding/decoding can be achieved with high probability with the three systems for different network configurations.

References

[1] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. F. Hu: Wireless Sensor Networks: a Survey on the State of the Art and the 802.15.4 and ZigBee Standards. In: Computer Communications (2007), doi:10.1016/j.comcom.2006.12.020.

[2] G.N.C. Kirby, A. Dearle, R. Morrison, M. Dunlop, R.C.H. Connor, P. Nixon: Active Architecture for Pervasive Contextual Service. In: International Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC 2003), ACM/IFIP/USENIX International Middleware Conference (Middleware 2003), Rio de Janeiro, Brazil, pp. 21-28.

[3] Karp, B., Kung, H.T.: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In: Proc. of MobiCom 2000, Boston, (2000) 243–254

[4] Caruso, A., Chessa, S., De, S., Urpi, A.: GPS free coordinate assignment and routing in wireless sensor networks. In: Proc. IEEE Infocom'05, Miami, (2005)

[5] Ratnasamy S., Karp B., Shenker S., Estrin D., Govindan R., Yin L., Yu F.: Data-centric storage in sensornets with GHT, a geographic hash table. Mob. Netw. Appl. (MONET) **8**(4) (2003) 427–442

[6] Albano, M., Chessa, S., Nidito, F., Pelagatti, S.: Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks. Mobile Data Management (MDM) 2007, Mannheim, Germany, 2007.

[7] Albano, M., Chessa, S., Nidito, F., Pelagatti, S.: Data Centric Storage in Non-Uniform Sensor Networks. Proceeding of INGRID 2007, Santa Margherita Ligure, Italy, 2007.

[8] Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In: Proc. of MobiCom 2000, Boston, (2000) 56–67

[9] Crossbow Technology: Mica wireless measurement system datasheet In: www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA.pdf.

[10] Maddenand, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In: Proc. of the 2003 SIGMOD Conference, San Diego, (2003) 491–502

[11] L. M. Feeney, M. Nilsson: Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In: Proc. IEEE Infocom, Anchorage AK (April 2001)

[12] P. Santi and S. Chessa: Crash Faults Identification in Wireless Sensor Networks. In: Computer Communications, 25 (14) (2002), pp. 1273-1282

[13] R. Prakash and M. Singhal: Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems. In: IEEE Trans. On Parallel and Distributed Systems, pp.1035-1048, October 1996

[14] B. Yao, W. K. Fuchs: Proxy-based Recovery for Applications on Wireless Hand-held Devices In: Proc. IEEE Symposium on Reliable Distributed Systems, October 2000, pp. 2-10.

[15] F. Barsi and P. Maestrini: Error Correcting Properties of Redundant Residue Number Systems. In: IEEE Transactions on Computers, Vol. C-22 (3) (1973) 307-315

[16] D. Mandelbaum: Error Correction in Residue Arithmetic. In: IEEE Transactions on Computers, Vol. C-21 (6) (1972) 538-545

[17] Chessa, S., Di Pietro, R., and Maestrini, P.: Dependable and Secure Data Storage in Wireless Ad Hoc Networks: an Assessment of DS2 In: LNCS 2928, Proc. First Working Conference on Wireless On-demand Network Systems (WONS 04), Madonna di Campiglio (Trento, Italy) January 21-23 2004, pp.184-198.

[18] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos: A Prototype Implementation of Archival Intermemory. In: Proc. ACM 4th Conference on Digital libraries, Berkeley, CA, August 1999, pp. 28-37.

[19] J. Kubiatowicz et. Al.: OceanStore: An Architecture for Global-Scale Persistent Storage. In: Proc. ACM 9th Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, November 2000, pp. 190 201.

[20] S. Chessa and P. Maestrini: Dependable and Secure Data Storage and Retrieval in Mobile, Wireless Networks. In: Proc. IEEE Dependable Systems and Networks (DSN), San Francisco, USA, 22-25 June 2003.

[21] A. G. Dimakis, V. Prabhakaran, K. Ramchandran: Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes In: Information Processing in Sensor Networks (IPSN 2005), Los Angeles, California, USA, April 25-27 2005.

[22] I. M. Vinogradov: Elements of Number Theory In: New York: Dover, 1954

[23] Araujo, F., Rodrigues, L., Kaiser, J., Liu, C., Mitidieri, C.: CHR: a Distributed Hash Table for Wireless Ad Hoc Networks. In: Proc. of the 25th IEEE ICDCSW'05. (2005)

[24] Newsome, J., Song, D.: GEM: Graph EMbedding for Routing and Data-Centric Storage in Sensor Networks Without Geographic Information. In: Proc. of the First International Conference on Embedded Networked Sensor Systems, Los Angeles, (2003) 76–88

[25] Bian, F., Govindan, R., Schenker, S., Li, X.: Using hierarchical location names for scalable routing and rendezvous in wireless sensor networks. In: SenSys '04, New York, (2004) 305–306

[26] Shannon C.: A mathematical theory of communication. In: Bell Sys. Tech. Journal, 27:379-423, 1948

[27] N. Alon, H. Kaplan, M. Krivelevich, D. Malkhi, and J. P. Stern: Scalable secure storage when half the system is faulty. In: Automata, Languages and Programming, pages 576-587, 2000

[28] Berrou, C et. al.: Turbo Codes: General Principles and Applications. In: Proceedings of the 6th Tierrenia International Workshop of Digital Communications, Tierrenia, Italy, Sept. 1993.

[29] C.W.Hastings: Automatic detection and correction of errors in digital computers using residue arithmetic. In: 1966 Proc. IEEE Conference (Region Six), pages 429-464

[30] Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of the ACM **36**(2) (1989) 335–348