Dealing with Non Uniformity in Data Centric Storage for Wireless Sensor Networks

Michele Albano, *Member, IEEE,* Stefano Chessa, *Member, IEEE,* Francesco Nidito, and Susanna Pelagatti

***Abstract*—In-network storage of data in Wireless Sensor Networks (WSNs) is considered a promising alternative to external storage since it contributes to reduce the communication overhead inside the network. Recent approaches to data storage rely on Geographic Hash Tables (GHT) for efficient data storage and retrieval. These approaches, however, assume that sensors are uniformly distributed in the sensor field, which is seldom true in real applications. Also they do not allow to tune the redundancy level in the storage according to the importance of the data to be stored. To deal with these issues, we propose an approach based on two mechanisms. The first is aimed at estimating the real network distribution. The second exploits a data dispersal method based on the estimated network distribution. Experiments through simulation show that our approach approximates quite closely the real distribution of sensors and that our dispersal protocol sensibly reduces data losses due to unbalanced data load.**

***Index Terms*—Wireless Sensor Networks, Data Centric Storage, Information Dispersal, Load Balancing.**

I. Introduction

Wireless Sensor Networks (WSNs) [6] are a recent technology suitable for unattended monitoring of a wide range of environments, spanning infrastructures (such as factories or public buildings), houses or even humans. In a WSN, a set of low-power, inexpensive, embedded devices (called *sensors* or *nodes*) spontaneously cooperate to construct a wireless network to support their monitoring activities. Each sensor is a microsystem combined with a radio interface that embeds a set of transducers aimed at measuring different environmental parameters. A special sensor, called *sink*, acts as a gateway with the external networks, and it makes the sensed data available to external users.

In the early approaches, WSNs implemented an external storage scheme, for example using Directed Diffusion [12], where all sensed data are sent to the sink to be stored and analyzed outside the WSN. This scheme assumes that the sink has a permanent connection with the network, and that it performs most of the data analysis, while the role of the WSN is limited to data acquisition. However, the external storage approach is not feasible in applications where the WSN has an intermittent connection with the sink. For these reasons, the work described in [21] introduced the Data Centric Storage model with Geographic Hash Tables (DCS-GHT), in which data are stored within the WSN, for the sink to collect them at a future time, maybe after aggregation or pre-processing. Comparing this approach to the external storage approach, the authors observed that in-network storage contributes to save sensors' energy and to improve network lifetime. Since sensors have limited memory capacity, the storage of all the sensed data in the WSN may result impractical, however, with data centric storage it is possible to aggregate data thus reducing their size.

In this paper, we reconsider the DCS-GHT approach to data storage in WSN. In this approach, each datum is associated with a meta-datum; the meta-datum is hashed to a pair of coordinates (x, y) on the WSN area and the datum is then stored on the sensors forming a perimeter around (x, y). DCS-GHT constructs the perimeter by means of a geographic routing protocol [13], [9], [11]. We first deeply analyze the behavior of DCS-GHT through simulation. In particular, we analyze the effects of using a uniformly distributed hashing on non-uniformly distributed sensors and the effects of using perimeters comprising an unpredictable number of sensors. Our results show that, even on uniformly distributed sensors, the amount of stored data per sensor is extremely variable with DCS-GHT, which may lead to data losses in overburdened sensors. This phenomenon is even worse if the sensors are not uniformly distributed. For this reason, we introduce a novel approach, called Load Balanced Data Centric Storage (LB-DCS), to the storage of sensed data in a WSN. In our approach, sensors apply a distributed protocol to compute an approximation f of their actual distribution. Then f is used to bias the hash function in order to distribute coordinate pairs according to network distribution (more data stored on densely populated areas in the sensing field). Finally, a datum is replicated according to a QoS level that depends on the importance of the datum (as decided by the user). We evaluate thoroughly our approach using simulations based on NS-2 [19]. As compared with DCS-GHT our approach guarantees a much better load balancing of storage and greatly reduces the loss of data due to overburdened sensors. It should be stressed that, although our approach can be applied to network topologies which change dynamically due to sensors' movements or failures, our proposal is thought for networks with limited mobility. We are planning to extend this approach to WSNs with higher mobility in our future work.

The rest of the paper is organized as follows. Section II presents the related work and briefly describes DCS-GHT. Section III introduces the LB-DCS protocol and its mechanisms for density sampling, meta-data hashing, and data dispersal. In Section IV, we evaluate the performance of LB-DCS by means NS-2, and Section V draws the conclusions. Supplementary material is available at http://ieeexplore.ieee.org.

M. Albano, S. Chessa, F. Nidito and S. Pelagatti are with Dipartimento di Informatica, Università di Pisa, Pisa, Italy

M. Albano is also with Instituto de Telecomunicações (IT), Aveiro, Portugal

S. Chessa is also with Istituto Scienza e Tecnologia dell' Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy

II. Related Work

The *Data Centric Storage* (DCS) [21] defines a paradigm where data is stored within the network itself. In particular, each datum is associated to a meta-datum and the datum is stored in a set of sensors that is a function of the meta-datum.

The first proposal of DCS in WSN is with Geographic Hash Tables (DCS-GHT) [21]. In DCS-GHT, it is assumed that the geographic coordinates of sensors are known, and that each datum is described by a unique *meta-datum*. The set of sensors selected to store a datum is computed by means of a hash function applied to the corresponding meta-datum. This function returns a pair of geographic coordinates fitting in the area where the sensor network is deployed.

DCS-GHT exploits the primitive `put` for data storage and `get` for data retrieval. The `put` primitive takes in input a datum d and its meta-datum k. By hashing k, it produces a pair of coordinates (x, y) and it uses the GPSR routing protocol [13] to find the sensor closest to the coordinates (x, y) (called *home node*), and a set of sensors (called *home perimeter*) forming a perimeter around (x, y) (the details of GPSR are presented in Section II of the supplementary material). Then, to enforce data persistence against sensors' faults, the home node requires the sensors in the home perimeter store a copy of (k, d). The `get` primitive hashes the input parameter k (the meta-datum) to obtain the coordinate (x, y), then, by means of GPSR, it sends a request for the data with meta-datum k to the point (x, y). When this request reaches the sensors in the home perimeter around (x, y), they send back all the data they store that correspond to k.

Although innovative, DCS-GHT presents a number of limitations when deployed on real networks. It assumes a Uniform distribution of sensors and uniformly hashes meta-data on them. Moreover, if the WSN produces a large amount of data associated to the same meta-datum, all such data will be stored by DCS-GHT within the same home perimeter, thus overloading sensors on that perimeter. To avoid this problem DCS-GHT uses structured replication, which distributes data with the same meta-datum more evenly in the WSN [21]. However, as observed in our previous work [4] and discussed in Section III of the supplementary material, this is not enough to ensure load balancing. In fact, the storage load can become unbalanced even if meta-data are balanced and uniformly distributed.

Along this trend of research many alternative DCS mechanisms have been proposed. They are similar to DCS-GHT in the definition of the `put` and `get` primitives, but they differ in the internal mechanisms used to implement routing, data dispersal and storage. In particular *CHR* [5] organizes the WSN into clusters of sensors in order to address scalability issues related to routing and energy efficiency. *GEM* [18] constructs a labeled graph spanning the network to assign addresses to the sensors; this enables the routing of data by using such addresses rather than on geographical coordinates. *LHR* [8] bases routing on hierarchical location names of the sensors that are manually assigned. *RR* [23] associates the data to regions of the WSN rather than to single points in order to relax the requirements for location accuracy. *GLS* [14] provides cluster-based location services for locating data or nodes in grids and, even if not related to WSNs, it implements a geographic routing system relying on real-world geographic location information to route its queries. *DIM* [15] implements a geographic embedding of an index structure. It recursively divides the plane to assign addresses to sensors, then it hashes meta-data to that address space. *Comb-needle* [16] differs from the other approaches since it does not use meta-data. In Comb-needle, each datum is replicated on a number of sensors belonging to a vertical stripe of the WSN deployment area, and the retrieval is ensured since the queries are directed to the sensors belonging to a horizontal stripe of the same area. The positions and sizes of the stripes are optimized to ensure that data can be retrieved efficiently.

All these approaches neither consider non-uniformly distributed WSN, nor consider QoS and load balancing in the storage. The works in [7], [20] consider non-Uniform WSN, however their focus is on connectivity problems and broadcast protocols. Effects of non-uniformity in WSN data storage are taken into account in our preliminary works [2], [4]. In particular, [4] introduces a dispersal strategy that exploits a non-uniform hash function and that introduces the concept of QoS in the storage. However, it assumes that the network distribution is known a priori and it does not use any strategy to infer the actual distribution of the sensors. This fact is particularly limiting considering that the network distribution may change due to sensor faults. The works in [2] builds over [4] by introducing a primitive mechanism for the estimation of the density of the network. However this mechanism is intended only as a setup of the network and it is assumed that the network density does not vary with time. In this paper, we remove this assumption by introducing a mechanism for the on-demand estimation of the network density.

III. A Novel Protocol for Data Centric Storage

In our previous study [4], we observed that DCS-GHT is subject to load unbalance due to the fact that it relies on the home perimeters for data replication. Since the size of the perimeters can be extremely variable, this greatly affects the storage load of the sensors, that is also extremely variable. In Section III of the supplementary material, we show that this load unbalance results in consistent data leakages, both in Uniform and Gaussian distributed WSN.

This ill behavior of DCS-GHT is mainly due to three underlying assumptions:

- the density of the sensors is supposed to be known, hence DCS-GHT does not provide any means to inspect the network density;
- network density is supposed to be constant all over the WSN area, hence the hash function used by DCS-GHT to map meta-data to WSN area locations is a usual uniform hash function;
- there is no stress on load balancing, hence DCS-GHT selects all the sensors on the home perimeters for data storage, regardless of their size. Consequently, the set of sensors selected for the storage of a data can be very large or very small, without any means to control the size of this set.

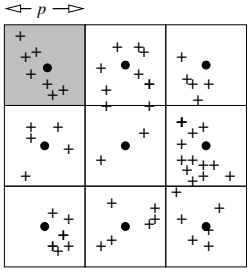


Fig. 1. WSN partitioning in squared regions.

The goal of coping with load balancing and non-Uniform WSNs led us to the design of Q-NiGHT [4], [3], that uses a generalized hash function to select the home node and that does not resort to a home perimeter to select the nodes that store data. Moreover, our seminal work [2] used a primitive technique to sample and distribute the actual WSN density. This section proposes a more organic approach, called *Load Balanced Data Centric Storage* (LB-DCS) to overcome the limitations of previous DCS systems. LB-DCS exploits three novel mechanisms to address the issues presented above. In particular:

- a density estimation protocol that provides the sensors with a network density estimation f to be used in the hashing function of meta-data in the `put` and `get` protocols;
- a hashing module that uses a generalized hash function biased with f;
- a storage protocol that enforces QoS in the selection of the sensors for data storage.

Summarizing, a `put` operation first acquires information on the actual sensor density in the WSN (as described in III-A), then it selects a home node in a load balanced way (as described in III-B), and finally it selects a set of nodes close to the home node, whose size is the QoS of the meta-datum at hand (as described in III-C). Similarly, a `get` operation first acquires information about the sensor density (as described in III-A), then it directs the query towards the home node of the meta-datum, which is computed using the same generalized hash function used by the `put` operation (as described in III-B).

A. Sampling density, and providing it to the sensors

This subsection describes how we can estimate network distribution and provide it to the sensors needing it. In general, we can have *static* networks, in which sensors do not move during the network lifetime and *dynamic* networks, in which density varies over time. For static networks, once density is computed it remains the same for all the network lifetime. In the case of dynamic networks, the lifetime of the network is divided into *epochs* of fixed time length. The density of the network is estimated at the beginning of each epoch and when an epoch expires all the density data are considered obsolete. The length of an epoch is system dependent and it may vary according to the network usage, to environmental conditions, to sensor mobility etc. In the rest of this section, we detail the protocol assuming a static network, intending that all the steps needed to perform the network density sampling should be repeated at the beginning of each epoch. On the other hand, each sensor uses a timer to understand when an epoch is finished and to discard outdated data accordingly.

To the purpose of estimating sensor density, the WSN is divided into $n \times n$ non overlapping square *regions* of side p (without loss of generality we assume that the WSN area is a square). The point at the center of a region is called *watch point*, and the sensor closest to a watch point acts as a *sentinel* for that region. An example of division of the WSN area into 9 regions is shown in Figure 1. Here, we spot with a black circle the center of each region (the watch-point) and with a "+" each sensor.

Note that p should be large enough to ensure that the sentinels are not in the radio range of each other, otherwise the same neighbor could be reported in two different regions. For large WSNs, p will be, in general, much larger than the sensors transmission range r.

The election of a sentinel in a region assures that the sentinel is closer than $r/2$ to the watch-point and it works as follows. First, each sensor computes its distance from the watch-point of the region where it belongs. This can be easily done if we assume each sensor knows the size of the WSN area and the side p of each region. Then, each candidate sentinel (i.e. each sensor closer than $r/2$ to the watch-point of its region) broadcasts its coordinates and its id to all its neighbors. As all candidate sentinels are within distance r, they all receive the coordinates and id of the other candidates and compute the closest one. If two or more candidates have the same distance from the watch-point the smaller id wins.

After the election, each sentinel broadcasts a request to its neighbors to count them. The number of neighbors is then used as an estimation of the local density in the region. Either proactive or reactive mechanisms can be used to deliver the estimates computed by sentinels. We consider one proactive protocol (*Broadcast*), and two reactive protocols (*Stripes* and *FatStripes*).

In Broadcast, each sentinel sends its estimate to all the sensors in the network. This is done once for all at the startup of each epoch.

In reactive protocols, when a sensor needs to perform a `put`/`get` operation it queries all the sentinels in the WSN to get their local density estimation. We can reach each sentinel by sending a message towards the watch-point of its region using GPSR. If the closest node is a sentinel it replies with its local density estimation; otherwise (i.e. if it is farther than $r/2$ from the region watch-point) it sends back a negative answer and the corresponding region is assumed to have local density equal to zero.

In Stripes, each sensor along the unicast route back from a sentinel caches the density estimation received (Figure 2). When a query for a sentinel arrives to a sensor, it first checks its cache. If an entry for that sentinel is present, the sensor sends back the cached density to the querying node without forwarding the query any further. Otherwise, it sends the request towards the sentinel using GPSR.

FatStripes is a step forward in optimization with respect to

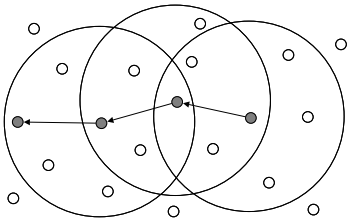


Fig. 2. Nodes that cache WSN density when Stripes is used.

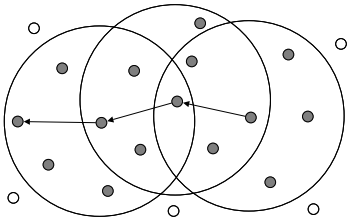


Fig. 3. Nodes that cache WSN density when FatStripes is used.

Stripes. As depicted in Figure 2, in Stripes only the relay nodes of GPSR cache the estimation received on the way back from a sentinel. However, all the nodes in the transmission range have actually spent energy to receive density information, even if they did not participate actively in the protocol. FatStripes uses this observation to cache density information also on the passive receivers as in Figure 3. We can observe that caching stripes are now larger than with Stripes, thus increasing the probability of a hit during a query. Note that the use of caches reduces the number of messages required in the protocol at the price of devoting some memory to cache density estimates on each sensor. If an estimate is stored in b bytes and recalling that n^2 is the number of sentinels, we need $n^2 \times b$ bytes of memory to cache estimates on each sensor. As the accuracy of the global estimate increases with the number of sentinels, for each application there is a trade-off between the memory used for caching and the accuracy required for density estimates.

Once a sensor has collected estimates for densities from all sentinels, it needs to figure out the density of the entire network. Here, we propose a simple algorithm to 'rebuild' density from samples.

The *rebuilding* algorithm acts in two steps. The first step computes an initial approximation for sensor density in each region. The second step refines the density approximation taking into account approximations in neighbor regions.

Since we have n^2 regions, we denote with w_{ij} the density estimate provided by the sentinel in region ij $(1 < i, j < n)$, and with d'_{ij} and d_{ij} the density approximations for region ij computed by the first step and by the second step of the algorithm, respectively.

The first step computes each d'_{ij} as

$$d'_{ij} = \frac{w_{ij}}{\sum_{ij} w_{ij}}$$

This first approximation works quite nicely if sentinels know a close estimate of the number of nodes in their region. However, it can be misleading at least in two opposite situations:

TABLE I
RejectionHash:

$h(k, f)$:
Requires: A key k and a function f.
Produces: A coordinate pair (x, y).

```
i = 0
while(true) {
  (x, y, z) = Hash(k + i)
  i = i + 1
  if z < f(x, y)
    return (x, y)
}
```

- *false zeroes:* if a region has got very sparse nodes near the watch-point and a concentration of many nodes along the border it can report a 0 or very low density which is not representative of the whole region.
- *over reporting:* if a region has got very sparse node near the borders and the majority of the nodes near the watch-point, it can report a much higher density than the real one.

To cope with these two problems, in the second step we compute the final approximation d_{ij} as a weighted mean of approximations computed in the first step for region ij and for its neighboring regions. The idea is to trust more d'_{ij} than the neighbor approximations, thus if region ij has m neighbors and we denote with N_{ij} the set of indices of neighbors regions, the second step computes the final approximation d_{ij} as:

$$d_{ij} = \frac{m * d'_{ij} + \sum_{i',j' \in N_{ij}} d'_{i'j'}}{2 * m}$$

The approximation of the network distribution is defined as the matrix $D = (d_{ij})_{n \times n}$ of the estimated distributions.

Relative merits of Broadcast, Stripes, and FatStripes and effectiveness of rebuilding algorithm are assessed in Section IV.

B. Load balancing the choice of the home node

This subsection describes the hashing module that is employed by the DCS system to map each meta-datum to the sensor that acts as the home node for it. Traditional approaches are prone to load unbalance, since standard hash functions distribute the home nodes uniformly over all the WSN area, and hence sensors in crowded regions are assigned a smaller number of meta-data.

On the other hand, once the network distribution is known, it is possible to use a generalized hash function to distribute home nodes, thus scattering data approximately with the same distribution of the sensors. This way the home node density is proportional to density of sensors in each region, and the mean number of meta-data assigned to each sensor tends to be balanced over the WSN.

A generalized hash function $h(k, f)$ is intuitively a pseudo-random number generator, that receives in input a probability distribution (the network distribution f) and a seed (the meta-datum k) and produces an output coordinate (x, y). The use of a pseudo-random number generator ensures that different keys are mapped w.h.p. on pairs that are not correlated to each other.

The pseudo-code for the hash function used in this module is described in Table I. It uses a strategy similar to the one used in the *rejection method* [17] to produce random numbers following any probability distribution in a limited domain. The probability function f is normalized such that the maximum value of f is 1, then a triple (x, y, z) is generated uniformly at random, (x, y) being a valid coordinate in the WSN area and $z \in [0, 1]$. If $z < f(x, y)$, the coordinate (x, y) is accepted and returned. Otherwise other triples (x, y, z) are generated until $z < f(x, y)$. The number of iterations of this method is in principle not limited, however in all of our experiments the method has ended in a few iterations.

Clearly, the probability of some value (x, y) to be returned by the generalized hash function is proportional to $f(x, y)$ (that is, to the network distribution in (x, y)), hence load balancing the number of assigned meta-data over the sensor nodes.

Note also that f can be obtained directly from matrix D. In particular, letting ij be the region that contains the point (x, y), f is obtained as $f(x, y) = d_{ij}$. Hence to store f is sufficient to store the matrix D of n^2 values, one for each region. The memory overhead of the sensor grows linearly with the number of watch-points (corresponding to the number of regions), because in the worst case each sensor has to cache the estimated density produced by each watch-point. However, as discussed in Section IV-A, the error in the density decreases rapidly with the number of watch-points, thus a limited number of regions (in our simulations around 25) is sufficient to attain a good trade-off between the precision of the protocol and the memory overhead.

C. Enforcing QoS in the storage

Similarly to DCS-GHT, LB-DCS is built atop GPSR and it offers the primitives `put` and `get` for data storage and retrieval. The `put` primitive takes three parameters: a datum d, its meta-datum k, and a QoS parameter q that expresses the level of dependability required for the datum d. The parameter q may be expressed using different metrics and ranges according to the particular redundancy technique used. In our case, since we use data replication as redundancy technique, q expresses the required number of replicas of the datum d. In particular q ranges in $[1, q_{max}]$ where q_{max} is the maximum number of replicas admitted. Note however that other redundancy techniques are also possible. For example, [1] investigates the use of erasure codes in data centric storage.

Let s be the source node of a `put(`d, k, q`)` operation. s first computes $(x, y) = h(k, f)$ as the destination of the packet $\texttt{P}_p$=<(x, y), <d, k, q>>. The packet in turn is sent to the destination using GPSR. As in DCS-GHT, we call *home node* the sensor H (of coordinates (x_H, y_H)), which is geographically nearest to the destination coordinates. Thus, H receives the packet as a consequence of applying GPSR. Upon the reception of packet $\texttt{P}_p$, H begins the *dispersal protocol*, which selects a set of q sensors (called the *replica set*) to store the replicas of (k, d). The replica set includes H and it is unique for a given key k.

The dispersal protocol is iterative and uses the concept of *ball*. Given the home node H of coordinates (x_H, y_H), we denote with $B_{(x_H,y_H)}(\overline{r})$ the *ball* centered in (x_H, y_H) of radius $\overline{r}$, that is the set of sensors that are within a Euclidean distance $\overline{r}$ from (x_H, y_H). Thus:

$$B_{(x_H,y_H)}(\overline{r}) = \{\text{sensors of coordinate}(x, y) : |(x_H, y_H), (x, y)| < \overline{r}\}$$

Sensor H then sends a request for storage to all the sensors within the ball. In turn, when a sensor in the ball receives the request for storage of $\texttt{P}_p$, it acknowledges the request to H. H accepts the $q - 1$ acknowledgments received from the nearest sensors, it confirms them, and disregards the others. The confirmation requires an extra packet sent by H. Sensors receiving the confirmation keep the data while the others disregard them after a timeout. If H receives $q' < q$ acknowledgments, then it executes another iteration of the dispersal protocol with $\overline{r} = 2\overline{r}$ in which it considers only the sensors in $B_{(x',y')}(2\overline{r}) - B_{(x',y')}(\overline{r})$. The dispersal protocol stops as soon as q sensors have been hired or the outermost perimeter has been reached. In the special case where at least $q - 1$ sensors are in the transmission range of H, H computes the set of the sensors in $B_{(x_H,y_H)}(\overline{r})$, inserts their identifiers into the request for storage, and only these sensors acknowledge the request for storage and store the datum; this way, not only the dispersal protocol performs only one iteration, but the number of acknowledgments generated is limited to $q-1$. It can be observed that our dispersal protocol is based on a geo-multicasting protocol [22].

The complexity of the `put` protocol clearly depends on the choice of $\overline{r}$ as this determines the number of iterations made to successfully place the q replicas. However, since we know the distribution of sensors f, for any given home node H of coordinate (x_H, y_H) and q it is possible to fix $\overline{r}$ in such a way that, with high probability, at least q sensors belong to the ball $B_{(x_H,y_H)}(\overline{r})$.

When a sensor s' of coordinates (x', y') executes `get(k)`, it first computes $(x, y) = h(k, f)$, and sends a query packet $\texttt{P}_g$=<(x, y), <$(x', y'), k$>> by means of GPSR. If the network topology is not changed since the execution of the `put` for the same meta-datum (i.e. the sensors have neither moved nor failed) then GPSR guarantees that the query packet will eventually reach the sensor closest to (x, y), i.e. the sensor H that was selected by the `put` protocol. Note however that in some cases the packet may reach another sensor in the replica set before reaching H. In any case, either H or the sensor in the replica set of k receiving the query packet will respond to s' by sending back all the stored data that match with the meta-data k.

However, if some sensors have failed, GPSR may fail to reach H and the other sensors in the replica set. Nevertheless, LB-DCS can ensure the retrieval of a data stored with QoS parameter q provided that up to $q - 1$ sensors in the WSN have failed and that the WSN is still connected (and hence the success of the `get(k)`). In particular, we analyzed the cases which may prevent the retrieval of a data stored with QoS parameter q when up to $q - 1$ sensors have failed, and, by simulation, we proved that these cases are extremely

unlikely. This result is discussed in detail in Section VI of the supplementary material.

Note that the capability of LB-DCS to retrieve stored data is also affected by sensors' mobility that may significantly change the network topology. To deal with mobility we consider the use of a Periodical Refresh Protocol (PRP) similar to the one introduced in [21]. Section IV of the supplementary material discusses the application of PRP to LB-DCS and its performance.

IV. Simulation

This section presents the simulation results on the cost and the performance of the density sampling protocols (Subsection IV-A), on the performance of the rejection hash technique in the dispersal of data in WSN with Uniform and Gaussian distributions (Subsection IV-B) and on the cost of the `put` and `get` protocols of LB-DCS and DCS-GHT (Subsection IV-C). To this purpose we have implemented both LB-DCS and DCS-GHT in the NS-2 simulator [19].

A. Simulations on density sampling

This section presents the simulation results on the error in the approximation of the network density computed by LB-DCS and on the cost of Broadcast, Stripes and FatStripes protocols.

For each simulation run, 100 WSNs are randomly generated, and the three protocols (Broadcast, Stripes and FatStripes) are run on them to evaluate the error in the density estimation and the messages exchanged by these protocols. The simulation iterates the runs until the outputs of the simulator reach a 99% confidence interval that is less than 1%. We report here the main results obtained with 200 sensors with a transmission range of 25m in a WSN area of $100 \times 100m^2$, and network density (expressed as the average number of neighbors per sensor) equal to 39. Note that we obtained similar results for different sizes of the WSN area and/or network density.

In Figure 4, we show the errors measured by approximating a Uniform, Gaussian and Hill distribution when varying the number of regions used. We measure the error using the Mean Square Error (MSE), a scalar quantifying the distance between the real distribution and the distribution computed by the rebuilding algorithm. If we denote with $D^R = (d^R_{ij})_{n\times n}$ the matrix of real region densities (and we recall that $D = (d_{ij})_{n\times n}$ is the matrix of the estimated region densities), the MSE is defined as the average of the square of the errors on each region, $MSE(D) = \frac{\sum_{ij}(d_{ij}-d^R_{ij})^2}{n^2}$. The error is under 0.0035 with 9 regions and falls under 0.0001 with 25 regions. A detailed evaluation of the error due to the rebuilding of a Gaussian distribution with 10×10 regions is given in Section V of the supplementary material.

Figure 5 compares the behavior of Broadcast, Stripes, and FatStripes in terms of the average number of messages generated, with respect to different numbers of sensors that query the sentinels. Since Broadcast is proactive, its performance is independent of the number of sensors' requests: all the sensors receive the sentinels' data ahead of time, so no real "request" is generated. Stripes has a good behavior when a small number of sensors ask for WSN density, but it suffers from its "unicast" communication when most of the sensors request this information. On the other hand, FatStripes gets the best of both worlds, since it is reactive and hence it is cheap when a few sensors query the sentinels, but it fully exploits the broadcasting characteristic of the physical medium to disseminate density information as much as possible. Hence, when many sensors request density information, most of them already have it because of the communication performed by past requests.

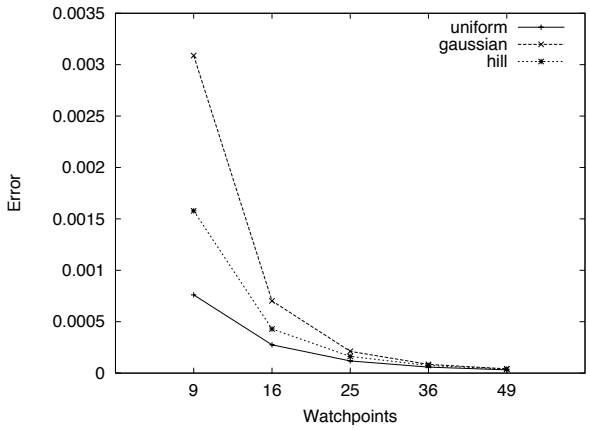


Fig. 4. Mean Square Error of estimated distribution (a scalar) measured for different values of p.

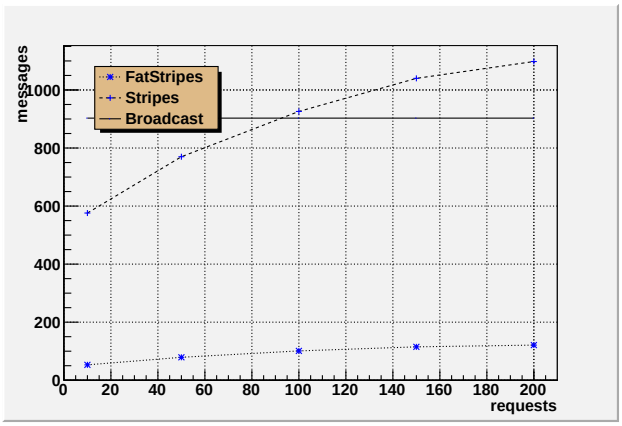


Fig. 5. Number of messages sent, against number of sensors that requested WSN density, $100 \times 100m^2$ WSN area, 200 nodes.

A detailed evaluation of the number of sent and received messages of Broadcast, Stripes, and FatStripes can be found in Section V of the supplementary material.

B. Simulating rejection hash

The simulation results presented in this section are aimed at evaluating the data leakage of LB-DCS and to compare it with the data leakage of DCS-GHT, which is analyzed in detail in Section III of the supplementary material. Data leakage occurs when a sensor serves too many `put` operations and it saturates its memory. In the simulations, we consider WSNs deployed in a square area of $200 \times 200m^2$, a transmission range of $10m$, and network densities (expressed as average

number of neighbors per sensor) in the range $[7, 30]$. The simulations consider both Uniform and Gaussian distribution of the sensors. In the case of LB-DCS, we also set the QoS parameter $q = 7$.

We assume sensors with storage capacity of $512KB$ (as it is the case of the Crossbow Mica family [10]). For each network generated in the experiments, the simulator executes a number of `put` operations for each sensor, each one accounting for 8 bytes (i.e. each `put` operation requires the storage of 8 bytes of data in each sensor in the home perimeter or in the replica set). In these simulations, we assume that each sensor produces a total of $2,100$ `put` operations during its lifetime, i.e. each sensor produces an amount of data to be stored that corresponds approximatively to $1/30$ of its memory capacity. We assume that, once a sensor is requested to store a new datum but its memory is full, it drops an older datum. This means that once a sensor reaches its maximum storage capacity it starts dropping data whenever it is requested to store new data.

With these settings, we performed the following experiment. In each simulation run, the simulator generates a new network (according to the chosen network distribution), it simulates all the `put` operations, and it computes the number of sensors that do not leak data and the total quantity of data lost. The simulation iterates until the average number of sensors that leak data and the average number of lost data reaches a 99% confidence interval that is less than 1%.

The result of these simulations is that with LB-DCS a negligible fraction (less than 0.00001) of the nodes leaks some data, and that a negligible fraction (less than 0.00001) of the data is lost. On the contrary, the fraction of data lost by DCS-GHT is around 0.7 (more details are available in Section III of the supplementary material and in [4]). This means that, under this respect, LB-DCS significantly outperforms DCS-GHT.

Further simulations were thus performed to understand the reasons of this behavior. The additional simulations evaluate the average load of the sensors in WSNs with network density equal to 14 and both Uniform and Gaussian distribution. Figure 6 and 7 report, for different values of the load on the x axis, the average number of sensors that have that storage load in the cases where the sensors are distributed according to a Uniform distribution or to a Gaussian distribution, respectively. The load on the x axis represents the number of data stored by a sensor normalized with respect to the data that a sensor produces. In particular, a sensor with load equal to 1 stores the same amount of data it produces (i.e. approximately $1/30$ of its memory capacity).

The results of the two cases are similar. In particular, DCS-GHT shows a great load unbalance (a significant number of sensors have a heavy load).

On the other hand, LB-DCS evenly distributes the load according to the estimated network distribution due to the rejection hash. In particular, with Uniform distribution most of the sensors have a limited load (around 7 data), and the average number of sensors that have the higher load (which in this case is of 17 data) is very small (about 0.05). Similarly, in the case of Gaussian distribution, most of the sensors have a limited load (around 8 data), and the average number of sensors with a high load (which in this case is of 18 data) is very small (about 0.5).

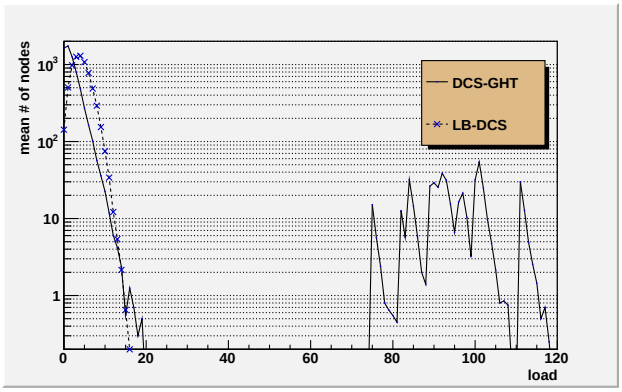


Fig. 6. Stored data per node for LB-DCS, Uniform distribution of sensors.

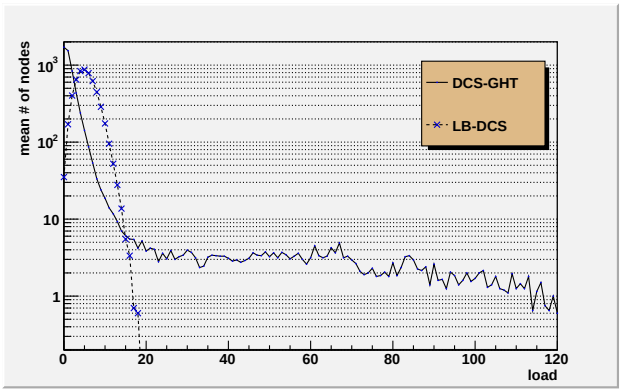


Fig. 7. Stored data per node for LB-DCS, Gaussian distribution of sensors.

C. Cost of the `put` and `get` operations

This subsection evaluates the cost of the `put` and `get` operations of DCS-GHT and of LB-DCS in terms of MAC layer send and receive operations.

In these simulations, we used the same parameters as in Section IV-B: WSN area of $200 \times 200m^2$, communication range of 10 meters, WSN density in the range $[7, 30]$, and QoS parameter of LB-DCS set to $q = 7$. Each simulation run iterates 1000 `put` operations, each followed by a `get` operation on the same meta-datum. The simulator reports the average number of packet forwarded and received by the sensors for each operation. The simulation runs are repeated until the simulator outputs reach a 99% confidence interval that is less than 1%. The simulator also reports on the correctness of each pair of `put` and `get` operations, in particular it computes the number of `get` operations that were unable to retrieve the values stored with the corresponding `put`. However, in our simulation experiments the `get` was always successful.

The simulation results on the cost of the `put` and `get` are reported in figures 8 and 9. In particular, Figure 8 reports the number of MAC-level send for the storage and retrieval operations of DCS-GHT and of LB-DCS, respectively, while

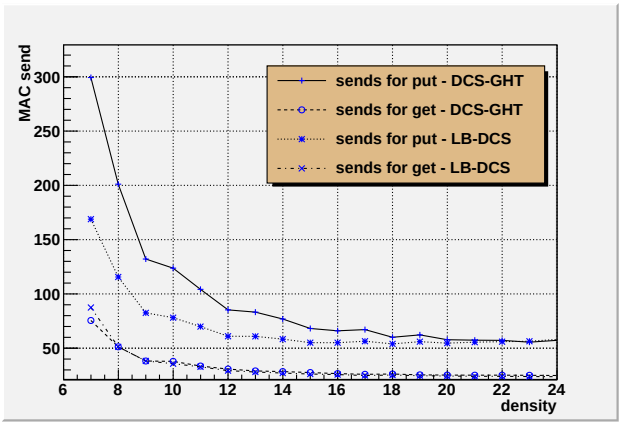


Fig. 8. MAC-level sends for `put` and `get` in DCS-GHT and LB-DCS.

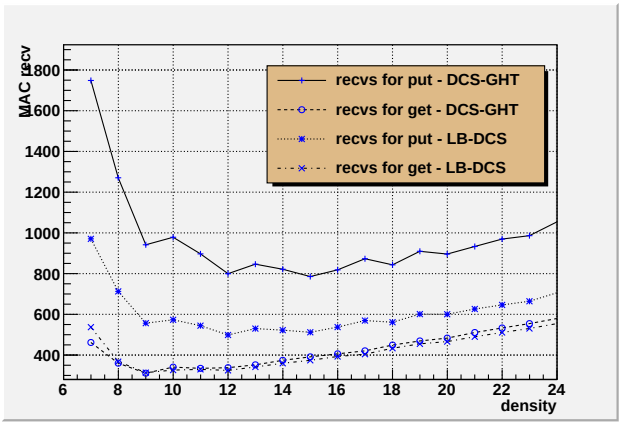


Fig. 9. MAC-level receives for `put` and `get` in DCS-GHT and LB-DCS.

Figure 9 reports the number of MAC-level receives. From the simulations, it results that the `put` operation is more expensive than the `get` operation. This happens because once the `put` request has reached the home node, it should also reach all the nodes in the home perimeter (in the case of DCS-GHT) or in the replica set (in the case of LB-DCS), which can be quite large depending on the DCS scheme used and on the desired redundancy of the data. Furthermore, it is seen that the `put` operation is less expensive with LB-DCS than DCS-GHT, due to the smaller cost incurred by the dispersal protocol, while the cost of `get` is almost the same for the two protocols.

V. Conclusions

DCS systems are very effective in implementing an in-network data storage and retrieval system, since they require only unicast communications. However, existing approaches disregard issues related to load balancing of the sensors and QoS. In this paper, we have addressed such aspects by proposing a new DCS system, LB-DCS, that relies on three mechanisms: a network density estimation protocol, a rejection hashing technique that produces pairs of coordinates by taking into account the real network distribution, and a dispersal protocol that enforces QoS and load balancing. The simulation results show that our approach significantly balances the storage load on the sensors and it adapts to different network distributions. As future directions, we believe that having the opportunity of estimating on the fly the distribution of the sensors may be exploited also in a better balancing of the routes and in the evaluation of the coverage of the sensing tasks. Future work includes also the extension of our approach to release the assumption of limited mobility of the sensors.

References

[1] Albano, M., Chessa, S.: Distributed Erasure Coding in Data Centric Storage for Wireless Sensor Networks. In: 14^{th} IEEE Symp. on Computers and Communications (ISCC),Tunisia, 67–75 (2009)

[2] Albano, M., Chessa, S., Nidito, F., Pelagatti, S.: Data Centric Storage in Non-Uniform Sensor Networks. In: Grid-Enabled Remote Instrumentation, eds. Davoli, F., Meyer, N., Pugliese, R., and Zappatore, S. Springer Series on Signals and Communication Technology, ISBN 978-0-387-09662-9, 3–19 (2009)

[3] Albano, M., Chessa, S., Nidito, F., S.Pelagatti: Q-night: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks. Technical report 06-16, Dipartimento di Informatica, Università di Pisa (2006)

[4] Albano, M., Chessa, S., Nidito, F., Pelagatti, S.: Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks. In: 8^{th} Mobile Data Management (MDM), Germany, 166–173 (2007)

[5] Araujo, F. et. Al.: CHR: a Distributed Hash Table for Wireless Ad Hoc Networks. In: 25^{th} IEEE ICDCSW. (2005)

[6] Baronti, et. Al.: Wireless Sensor Networks: a Survey on the State of the Art and the 802.15.4 and ZigBee Standards. Computer Communications, **30** (7), 1655 – 1695 (2007)

[7] Bettstetter, C.: The Cluster Density of a Distributed Clustering Algorithm in Ad Hoc Networks. In: IEEE ICC, Paris, 4336–4340 (2004)

[8] Bian, F., Govindan, R., Schenker, S., Li, X.: Using Hierarchical Location Names for Scalable Routing and Rendezvous in Wireless Sensor Networks. In: 2^{nd} ACM SenSys, Baltimore, 305–306 (2004)

[9] Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. Wireless Networks, **7**(6), 609–616 (2001)

[10] Crossbow Technology: http://www.xbow.com

[11] Frey H., Stojmenovic I.: On Delivery Guarantees of Face and Combined Greedy-Face Routing in Ad Hoc and Sensor Networks. In: ACM MobiCom, Los Angeles, 390-401 (2006)

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

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

[14] Li, J., Jannotti, J., De Couto, D. S. J., Karger, D. R., and Morris R.: A Scalable Location Service for Geographic Ad hoc Routing In: ACM MobiCom, Boston, 120–130 (2000)

[15] Li, X., Y.J. Kim, R. Govindan, W. Hong: Multi-Dimensional Range Queries in Sensor Networks. In: 1^{st} ACM SenSys, Los Angeles, 63–75 (2003)

[16] Liu, X., Q. Huang, Y. Zhang: Combs, Needles, Haystacks: Balancing Push and Pull for Discovery in Large-Scale Sensor Networks. In: 2^{nd} ACM SenSys, USA, 122–133, (2004)

[17] Neumann, J. V.: Various Techniques Used in Connection With Random Digits. In: Taub, A. H., editor, John von Neumann, Collected Works, volume 5, pages 768–770. Pergamon Press, Oxford (1951)

[18] Newsome, J., Song, D.: GEM: Graph EMbedding for Routing and Data-Centric Storage in Sensor Networks Without Geographic Information. In: 1^{st} ACM SenSys, Los Angeles, 76–88 (2003)

[19] Network Simulator 2 (ns-2): http://nsnam.isi.edu/nsnam/

[20] Orecchia, L., Panconesi, A., Petrioli, C., and Vitaletti, A.: Localized Techniques for Broadcasting in Wireless Sensor Networks. In: DIALM-POMC '04, New York. ACM Press (2004)

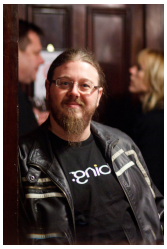
[21] Ratnasamy S., Karp B., Shenker S., Estrin D., Govindan R., Yin L., Yu F.: Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table. MONET **8**(4), 427–442 (2003)

[22] Seada, K., Helmy, A.: Efficient and Robust Geocasting Protocols for Sensor Networks. Computer Communications **29**(2), 151–161 (2006)

[23] Seada, K., and A. Helmy: Rendezvous Regions: A Scalable Architecture for Service Location and Data-Centric Storage in Large-Scale Wireless Networks. In: 4^{th} WMAN, satellite of IEEE/ACM IPDPS, New Mexico (2004)

Michele Albano received his BSc degree in Physics in 2003, and his BSc, MSc and PhD degrees in Computer Science in 2004, 2006 and 2010 respectively, all of them from the Un iversity of Pisa, Italy. He was visiting researcher at Universidad de Malaga in 2007, at Stony Brook University in 2009, and before being a researcher he worked as software engineer and wireless technologies specialist in private companies in the period 2001-2006. In 2006 and 2007 he was involved in EU funded projects SMEPP and XtreemOs, and he is now a postdoc researcher at the Instituto de Telecomunicações - Pólo de Aveiro, Portugal, working on EU funded projects C2POWER and PEACE. He has co-authored more than 20 papers published on international journals and conference proceedings, he is reviewer for several international journals and conferences. His main research interests are in the areas of wireless networks and peer-to-peer networks.

Stefano Chessa received his MSc and PhD degrees in Computer Science from the University of Pisa, Italy, in 1994 and 1999, respectively. Since 2000 he is Assistant Professor at the Computer Science Department of the University of Pisa and since 2003 he is also Research Associate at the ISTI/CNR Institute (Information Science and Technology Institute). He has been involved in many national and European projects. In particular he has coordinated the CNR project Management of Data in Wireless Sensor networks (MaD-WiSe). He has also participated to the EU FP6 SatNex, SMEPP, InterMedia, PERSONA projects and to the EU FP7 universAAL project. He has co-authored more than 80 papers published on international journals and conference proceedings, he is reviewer for several international journals and conferences, and he has been member of several international program committees of conferences and workshops. His research interests are in the areas of wireless ad hoc networks, a nd wireless sensor networks, security in wireless communications, and system-level diagnosis.



Francesco Nidito received his PhD in 2008 from the University of Pisa, Department of Computer Science. He performed his research activity at the same University and as a visiting scholar at the Northeastern University, Boston (MA) collaborating with Prof. Stefano Basagni and Andras Farago. After the Ph.D., he has been working in the search engines field, first at Ask.com and then at the Microsoft Search Technology Center of London (UK).

Susanna Pelagatti received her MSc and PhD degrees in Computer Science from the University of Pisa, Italy,in 1987 and 1993, respectively. In 1995, she joined the University of Pisa, Dipartimento di Informatica as an Assistant Professor. Since 2002 she is an Associate Professor in the same department. Her main research interests are in the areas of parallel computing, wireless and ad Hoc networks, and sensor networks. She has been involved in national and international projects and she has co-authored more than 50 papers published in international journals and conference proceedings. She has been member of international program committees of conferences and workshop.