Data Centric Storage in Non-Uniform Sensor Networks

Michele Albano[1], Stefano Chessa[1,2], Francesco Nidito[1], and Susanna Pelagatti[1] *

[1] Computer Science Department, University of Pisa, Largo Pontecorvo 3, 56127 Pisa, Italy
[2] Istituto di Scienza e Tecnologie dell'Informazione, Area della Ricerca, CNR di Pisa, S.Cataldo, 56100 Pisa, Italy

Summary. Dependable data storage in wireless sensor networks is becoming increasingly important, due to the lack of reliability of the individual sensors. Recently, data centric storage (DCS) has been proposed to manage in network sensed data. DCS reconsiders ideas and techniques successfully proposed in peer to peer systems within the framework of wireless sensor networks. In particular it assume that data are uniquely named and data storage and retrieval is achieved using names instead of sensor nodes addresses. In this paper, we discuss the limitations of previous approaches, and in particular of Geographic Hash Tables (GHT), and introduce DELiGHT, a protocol which provides fine QoS control by the user and ensures even data distribution, also in non uniform sensor networks. The merits of DELiGHT have been evaluated through simulation in uniform and Gaussian distributed sensor networks. The simulation results show that the protocol provides a better load balancing than the previous proposals and that the QoS is ensured without appreciable overhead.

1 Introduction

A sensor is a micro-system which comprises a processor, one or more sensing units (transducers), a radio transceiver and an embedded battery. A Wireless Sensor Network (WSN) is a network formed by a large number of tiny and inexpensive sensors that cooperate to perform measurement tasks [1]. Sensors are spread in an environment (*sensor field*) without any predetermined infrastructure and cooperate to execute common monitoring tasks which usually consist in sensing environmental data. The sensed data are collected by an external sink node, when it is connected to the network. The sink node,

* This work was supported in part by the 6th EU framework programme projects SMEPP (contract number 033563), PERSONA (contract number 045459), and INTERMEDIA (contract number 38419).

which could be either static or mobile, is in turn accessed by the external operators to retrieve the information gathered by the network. In a WSN, data can be accessed according to different paradigms which define sensor-to-sensor and sensor-to-sink communications. Recent paradigms [2–4] see the WSN as a distributed database. The WSN is then "programmed" by the sink by sending queries to the sensors. Sensors in turn reduce and filter the sensed data locally before sending an answer.

In this scenario, dependable innetwork data storage is becoming increasingly important, due to the lack of reliability of sensor nodes [5], [11], [12], [13].

In WSNs the content of data is generally more important than the address of the sensor that has gathered the data. This naturally lead to Data Centric Storage (DCS) [5], in which data is univocally named and the node in which data are stored is determined by the name associated with them.

GHT [5] implements Data Centric Storage using Geographic Hash Tables. Each datum has a unique *meta-datum* (or name) which is hashed uniformly as a coordinate in the sensing area, represented as a two-dimensional plane. GHT implements two operations: `put`, which stores data, and `get`, which retrieves them. In the `put` operation, the name of data to be stored is first hashed into a location (x, y) in the sensing field. Then, GHT selects the closest sensor to (x, y), which becomes the *home node* for that data. The home node is selected using GFG routing protocol [6]. GFG uses two operation modes: greedy and perimeter. Each packet starts in the *greedy mode*, in which it is routed progressively closer to its destination at each hop. When a packet reaches a node s_i whose neighbors are all farther than s_i to the destination, GFG switches to the *perimeter mode* and the packet is forwarded using the *right hand rule* (the packet is forwarded on the *next* edge clockwise from the edge from which the packet has been received). As soon as the packet reaches a node closer to destination than the previous ones, it returns to the greedy mode. If the destination (x, y) does not correspond to any sensor, GHT uses the perimeter mode of GFG to locate all the sensors surrounding (x, y) (called the *perimeter* of (x, y)). The closest sensor in the perimeter becomes the *home node* for (x, y). GHT stores a copy of the data in the home node as well as in all the sensors belonging to the perimeter. Storing on all the perimeter is essential to guarantee data persistence also in presence of node faults. Data retrieval uses a `get` operation. The name is first hashed into the destination (x, y), then GFG is used to route the request to (x, y). When the request reach a node in the perimeter of (x, y), the data is returned back to the sender. Replicating all data on the perimeter of (x, y) is a simple choice, which allows to use GFG with almost no changes and which can work quite well on very large sensing fields with uniformly distributed sensors. However, this choice has two important drawbacks. The first one is that in non uniformly distributed sensor networks the number of data stored on each sensors can be very different. This lead to load unbalance (a few nodes need to answer many gets) and to rapid consumption of sensor batteries. The second problem is

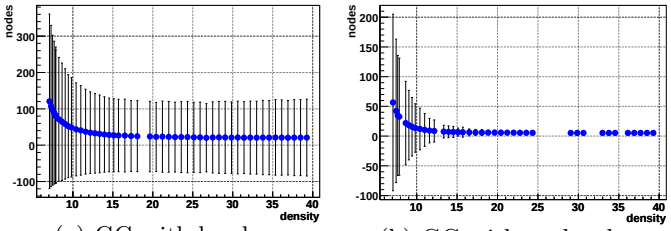
related to dependability and QoS. In GHT, the user has no mean to ask for better dependability for a really important datum. This is because the number of replicas of each datum depends only on the GFG perimeter it happen to be hashed its home node.

Other data centric storage for sensor networks [11], [12], [13] differ from GHT in terms of algorithms used to compute the set of sensors which should store a given data, but as GHT they fail to address QoS in data dependability and they do not address issues related to sensors distribution.

Our work originates form GHT. In some previous work, we proposed QNiGHT [7, 8], in which we implemented QoS using geographic hash functions. In QNiGHT, a user can specify an extra parameter for put operations which tell the system the level of QoS (ie, the number of replicas) required for a given datum. Then, the protocol takes care of spreading exactly this number of replicas in the surroundings of the home node. Results show that QNiGHT was actually able to provide QoS with small extra overhead with respect to original GHT puts and gets. However, to reach these results, QNiGHT needed to know exact sensor distribution function in advance, showing poor results and load unbalance when the actual sensor distribution was different.

In this paper we propose DELiGHT (Density Estimation for Load balancing in Geographic Hash Tables), a novel DCS protocol which moves from GHT and QNiGHT in order to provide QoS control and good load balance among sensors in WSNs of any (possibly changing) distribution. DELiGHT uses a strategy similar to the *rejection method* [9] to build a hash function biased with sensor distribution. This spreads data more evenly among nodes. In addition, DELiGHT can provide QoS with different redundancy techniques. We detail the protocol using pure replication, allowing the user to choose the number of replicas required for a given datum. We conduct detailed simulations of DELiGHT, QNiGHT and GHT and compare the results obtained with respect to the load of each sensor (i.e. the number of data stored in each node) and the number of messages needed for data storage and retrieval. Results show the good performance of DELiGHT on different sensors distributions on terms of both protocol costs and load balance.

The paper is organized as follows. Section 2 discusses more in details the problems of load unbalance in GHT presenting the results of simulations carried on with uniform and Gaussian distributions. Section 3 presents the density estimation technique used in the protocol. Then, Section 4 details DELiGHT and discusses how different redundancy techniques can be incorporated in the protocol. Finally, Section 5 reports on the simulation of our protocol and compares its performance with plain GHT. Conclusions are presented in Section 6.



(a) GG with border (b) GG without border

Fig. 1: Mean and variance of perimeters (number of nodes) measured for different densities with GG planarization.

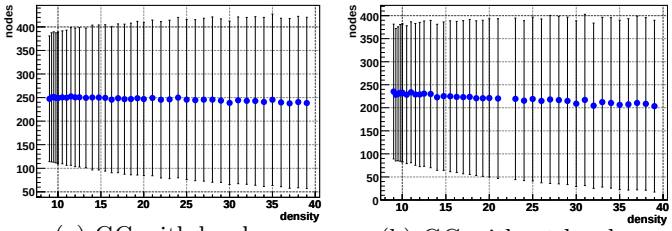
2 Load unbalance in GHT

GHT [5] implements Data Centric Storage using Geographic Hash Tables and the GFG protocol. We have already discussed how the protocol works in the introduction. Here, we discuss the behaviour of GHT, w.r.t. load balancing. In particular, we describe a simulation experiments with gives better insigths on the protocol behaviour and settle some motivations for our proposal.

Our experiment is organized as follows. In order to measure the degree of unbalance of GHT on realistic scenarios, we simulated a flat square sensing field, with a $400m$ side. Each node has circular transmission range with $10m$ radius. In this area, we simulated several WSNs ranging from 3600 to 20000 sensors, which correspond to a mean network density ranging in $[8, 40]$. For each density, we randomly generated 100 networks with uniform distribution. For each network, we compute the mean and the variance of the number of nodes found in a GFG perimeter as follows. For each sensor network the simulator randomly selects 1000 points and, for each point, it computes the number of nodes in the perimeter surrounding the point. GFG need to work with a planar graph in order the perimeter mode to behave correctly. We assume to planarize graphs using the GG (*Gabriel Graph*) [10] algorithm. Figure 1.a shows that, as the network density increases, the average number of nodes in a perimeter decreases. However, the actual number of nodes remains highly variable. This variance is partly due to the behavior of nodes in the outer part of the sensing area, since in that area the probability of having very long perimeters (i.e. following the whole boarder) is high. With low densities, the probability that a random point belongs to the exterior of the network (and thus it is associated to the external perimeter) is not negligible.

Perimeters on the border.

In order to understand this *border effect*, we performed another set of simulations in which the sensing networks are generated in the same way as above but the external area is not used to store data. We "cut away" the 5% of the area from each border, for a total of 19% of the total area. We randomly



(a) GG with border (b) GG without border

Fig. 2: Mean and variance of GHT perimeters for different WSNs densities, Gaussian sensor distribution.

generate 1000 points in the white area and again measure the length of each perimeter and compute the mean and the variance. Figure 1.b shows the results we computed. The mean and the variance improve if the border nodes are let out but standard deviation remains high, leading to a high load unbalance in the nodes.

Non uniform sensor distribution.

In order to understand the behavior of GHT with non-uniform sensor distribution, we repeated our experiments using a Gaussian function ($\sigma = 1$ with maximum on the center of the area) for distributing sensors. The function is structured to have the 99 percentile matching the area. The results are shown in Figure 2. The behavior is much worse than with uniform distribution because GHT, uses a uniform hash function independently of the real distribution of the sensors. This bring to a pathological state of *load unbalance* that is due to the different quantity of data that must be managed by an equal number of sensors: A sensor on the border of the deployment area must manage a quantity of data that is larger than the quantity managed by a sensor in the center of the network.

Load unbalance and QoS.

Another issue with GHT is that there is no way to control the QoS provided for each datum. Since the point (x, y) is obtained computing an hash function h on its associated meta-data M, the selection of the sensors candidate for storage is in practice independent from the importance of the datum. In principle this ensures the same treatment for each stored datum. However, if the meta-datum M is particularly popular and many sensors generate data described by M, the sensors located in the perimeter around $(x, y) = h(M)$ would be burdened with an high storage and communication load. For this reason the authors of GHT[5] introduce the technique of *structured replication*, that replicates the same datum in different sub-areas. However nor GHT, neither the structured replication ensure that the level of redundancy associated to a

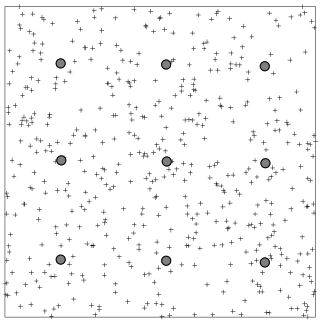


Fig. 3: Geographical points chosen for 9 *probes*.

data is related to the importance of the data itself: GHT assure only the same average treatment of each stored data. Another aspect is that, although the average level of redundancy of the meta-datum is constant, in practice it can vary significantly (due to the fact that each geographic points is surrounded by a different perimeter), even in case of uniform distribution of the sensors.

3 Distribution estimation

GHT is designed to work on uniform networks and the hash function is a plain uniform hash function. When the network is randomly distributed the sensors can be distributed in a non-uniform way. This may bring to a pathological state of load unbalancing. To ensure data load-balancing DELiGHT must be able to spread data following a function fitting the distribution of the sensors in the environment.

In DELiGHT this problem is addressed using a generic hash function (detailed in Section 4.2). This function takes in input two parameters: The first one is the meta-datum to hash and the second one is the probability distribution function with which we want to spread the data in the network.

A solution that uses the actual distribution of the sensors is impracticable because this distribution can be unknown or it may vary with time, and measuring the number of neighbours of each sensor and broadcasting the information is too expensive in terms of energy consumption. Thus, to use the non-uniform hash function technique, DELiGHT must be able to approximate the actual distribution of the nodes during the lifetime of the network.

The DELiGHT architecture exploits the measurement of a limited number of sensors (*probes*) using an *estimation function*, in order to approximate the actual distribution function of the nodes.

At network startup, the sink chooses a number of geographical points in the network and sends them an initialization message. The message is routed by GFG and the sensors which are the closest to each of the selected geographical

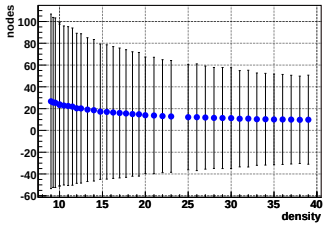


Fig. 4: Mean and variance of GHT perimeters for different WSNs densities, Gaussian sensor distribution and non uniform hashing.

points detect their number of neighbors. Then they store this information in the network using the `put` operation of DELiGHT under the assumption of uniform nodes' distribution, using `Probed Distribution` (PD) as key.

Every node that needs to estimate the distribution of the sensors does a `get` operation to retrieve the measurements. Then, the sensor uses the estimation function to calculate the distribution. In front of frequent topology modifications, it is possible for the *probes* to detect periodically their number of neighbors to update the estimation function. The tradeoff between the communication cost of the distribution of *probe*'s measurements and the fidelity of the estimation is evaluated in Section 5.

Probes' selection algorithm chooses the geographical points in a regular pattern, as shown in Figure 3. The number of *probes* is k^2, and the cartesian axis of the network area is divided in $k+1$ segments. All the segments on the x axis have the same length, except the first and the last which have half length. The y axis is divided in the same way. The geographical points are on the convergence of the delimiters of the segment. We set $k^2 = 25$ which proved a good tradeoff as shown in Section 5. Thus, the first and last segment of x axis is 10% of the x range, and each other segment is 20% of the range.

The estimation function is the weighted mean of the density measured by the 3 nearest *probes*, with the weight equal to 1 divided by the squared distance from the *probes*, or, in formulae, the density estimation for the geographical point (x, y) is

$$\rho(x,y) = \frac{\sum_{i=1}^{3} \frac{d_i}{(x-x_i)^2+(y-y_i)^2}}{\sum_{i=1}^{3} \frac{1}{(x-x_i)^2+(y-y_i)^2}} \tag{1}$$

with (x_i, y_i, d_i) respectively the coordinates and the number of neighbors of the 3 nearest *probes*.

Algorithm 1 `get(`K`)`

Require: A key K. PROBED is set to NULL during the initialization.
Ensure: Retrieves a datum D or null if K correspond to no data.

```
if PROBED = NULL OR K == PD then
  (x, y) ← RejectionHash(PD, UNIFORM)
  Route the request towards (x, y) using GFG.
  PROBED is set to the received PD
end if
if K != PD then
  // now PD is available
  (x, y) ← RejectionHash(K, PROBED)
  Route the request towards (x, y) using GFG.
  Return the received data associated to key K.
end if
```

4 DELiGHT: Extending Geographic Hash Tables with Density Estimation

In the initialization phase of DELiGHT the sink node selects the set of geographical points to be used to probe the network density and sends the initialization message to this points using GFG [6] as discussed in Section 3. Once the network density had been probed and stored in the network using the `put` operation of DELiGHT under the assumption of uniform distribution of the nodes, DELiGHT is full functional and can be used to store data by any node in the network taking into account the nodes' distribution.

To this purpose DELiGHT offers a modified version of the `put` and `get` operations. The first time a node performs either a `put` or a `get`, it preliminarily acquires PD (the `Probed Distribution`). The `put` and `get` operations are detailes in the next subsection.

4.1 The `put` and `get` operations

During the initialization, PROBED, a state variable which corresponds to meta datum PD is set to NULL. The `get` operation is used to retrieve a datum described by a metadata (which could also be PD itself). It takes as parameter a key (the metadata) and exploits the information about network distribution stored in PROBED, if available. If PROBED is NULL, then the `get` first acquires PROBED. This is achieved since the position of the nodes stored PROBED can be computed using the function `RejectionHash` over metadata PD and assuming uniform distribution of the nodes.

Otherwise, if PROBED is available, it directly computes the location of the desired data using the function `RejectionHash` over the input metadata and using PROBED as distribution of the nodes.

Algorithm 2 `put(k,D,Q)`

Require: A key K, a datum D, a QoS parameter Q. PROBED is set to NULL during the initialization.

Ensure: A datum D is stored in the network, with Q replicas.

```
if PROBED = NULL then
  PROBED ← get(PD)
end if
(x, y) ← RejectionHash(K, PROBED)
Route the request to the home node for (x, y) using GFG.
Store datum D using the dispersal protocol.
```

In DELiGHT the interface of the `put` includes, along with the meta-data K and the data D, also a parameter Q expressing the desired QoS. `put` uses PROBED, a state variable, to estimate the distribution of the sensors. If PROBED is not available, it is first retrieved using a `get` operation with meta-datum PD. Then the algorithm calculates the geographical coordinates (x, y) and it routes the request to the home node for (x, y). In turn the home node will apply the dispersal protocol, which is presented in Section 4.3. The dispersal protocol stores the datum D, using Q as a parameter of the dependability required for the data, in terms of number of replicas required for the data.

4.2 Non-uniform hashing

An hash function $h(k)$ is a kind of pseudo-random number generator: Starting from a seed (in our case the key k) it produces an output (in our case a value in R^2) such that for *near* values of the key, the hashed values must be *distant*. With this consideration in mind, we define a new hash function based on the *rejection method* [9], but with some differences. Rejection method is a technique used in random number generation to produce random numbers following any probability distribution, with limited dominion. The basic idea is the following. The probability function is boxed and we generate uniform random values in the box. If the value generated is below the distribution function, the value is accepted and returned. Otherwise we randomly generate new points in the box until a value is below the function. Notice that in principle there is a non-null probability of non termination because any number of subsequent values can be generated all above the function. In practice, a good uniform hash grants to generate values in all the box. The function `RejectionHash` returns a pair (x, y) of coordinates where to place data from its key `k`, belonging to the estimated PD. Instead of using uniform randomization, it uses random hashing on the key. At each iteration, if necessary, it changes lightly the key in a deterministic way. Figure 4 shows a good behavior of the non-uniform hash function. `RejectionHash` fits well the sensor distri-

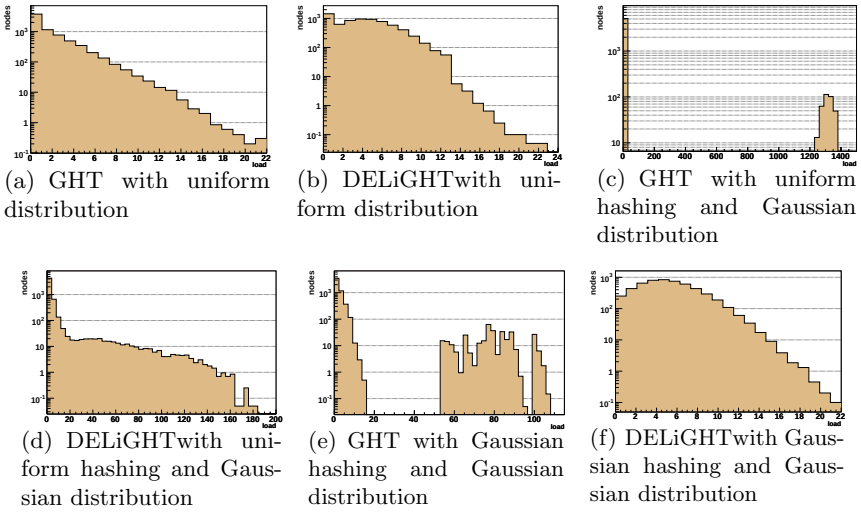
bution in the data dissemination strategy with a good global load balancing. These results are better than the results provided with uniform distribution and uniform hashing (Figure 1.a and Figure 1.b). This is due to the Gaussian distribution of the nodes that does not have a border effect as evident as in the uniform distribution.

4.3 The Dispersal Protocol

As in GHT we call *home node* the sensor s_d (of coordinates (x', y')) geographically nearest to the destination coordinates. The home node naturally receives the packet as a consequence of applying GFG. Upon the reception of packet $\mathtt{P}_p$, sensor s_d begins a *dispersal protocol* which selects Q sensors to store a copy of `<M,D>`. The dispersal protocol is iterative and uses the concept of *ball*. Given a sensor s_d of coordinates (x, y), we denote with $B_{(x,y)}(\overline{r})$ the *ball* centered in (x, y) of radius $\overline{r}$, that is the set of sensors that are within a Euclidean distance $\overline{r}$ from (x, y). In the first iteration s_d broadcasts a replica of `D` to all the sensors included in the ball $B_{(x',y')}(\overline{r})$. $\overline{r}$ is chosen in order to reach the Q sensors nearest to (x, y) with high probability. In simulations, to compute $\overline{r}$, we use a simple strategy based on the density estimation calculated as in Section 3. Using this heuristic, we have experimented that the method converges in no more than 2 iterations. Each sensor receiving a replica responds with an acknowledgment to s_d. Sensor s_d confirms the $Q-1$ acknowledgments received from the sensors geographically nearest to (x, y) and disregards the others. The confirmation requires an extra packet sent by s_d. Sensors which receive the confirmation keep the data while the other sensors will disregard the data after a timeout. If s_d 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. When a node s_g of coordinates (r, z) executes `get(M)` it firstly computes $(x, y) = h(M)$, and sends a query packet $\mathtt{P}_g$`=<(x,y),<(r,z),M>>` using the GFG protocol. In turn, packet $\mathtt{P}_g$ will reach the perimeter surrounding (x, y) and it will start turning around the perimeter. Eventually, the packet will reach either the home node or another node containing a replica of the data D associated to M. This node will stop packet $\mathtt{P}_g$ and will send the required data back to s_g. The complexity of the `put` protocol clearly depends upon the choice of $\overline{r}$ as this determines the number of iterations made to successfully place the Q replicas. However, if we know the distribution of sensors f, for any given (x', y'), coordinates of s_d, 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 $B_{(x',y')}(\overline{r})$.

4.4 Behavior in case of faults.

If some of the nodes holding the replicas of `<M,D>` fail, our protocol ensures graceful degradation of data availability. Due to GFG protocol, any `get` with



(a) GHT with uniform distribution

(b) DELiGHTwith uniform distribution

(c) GHT with uniform hashing and Gaussian distribution

(d) DELiGHTwith uniform hashing and Gaussian distribution

(e) GHT with Gaussian hashing and Gaussian distribution

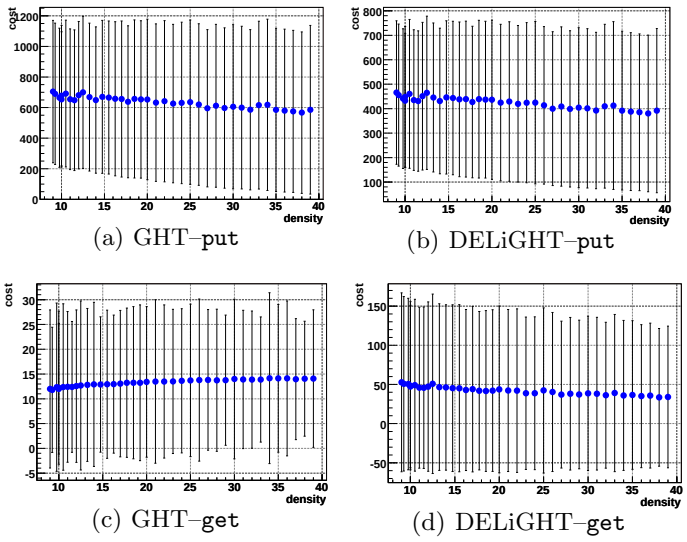
(f) DELiGHTwith Gaussian hashing and Gaussian distribution

Fig. 5: Average load of sensors

key `M` is routed to the node geographically nearest to $(x, y) = h(M)$ (home node of `M`). If the faulty node is not the home node, the protocol implicitly discards it. If this is not the case, the second nearest node in the perimeter is always included in the ball built by our protocol.

5 Simulations and results

In this section, we discuss the results of our simulation. We simulated a square with a $400m$ side, with sensor transmission range of a perfect $10m$ radius circle. We assumed a density of 14 and performed 2000 `put` operations with randomly generated meta-data using both GHT and DELiGHT˙ In these trials, DELiGHT uses a pure replication QoS with 15 replicas for each datum. Figure 5 compares the behavior of GHT (graphics a,c,e) and DELiGHT (graphics b,d,f). Charts (a,b) consider uniform sensor distribution and uniform hashing, (c,d) Gaussian sensor distribution and uniform hashing and finally (e,f) Gaussian sensor distribution and Gaussian hashing. In all graphs, the x axis shows the different load (e.g. Number of data) on a node and the y axis shows the number of nodes storing exactly this number of data. Values on the y axis follow a logarithmic scale for better comprehension. We can notice that DELiGHT reaches better load balance even in the uniform case (graphs a,b), while from graphs (c,d) it is evident that DELiGHT reaches better load balance even in the uniform case. Figure 5.(c,d) shows the average load of sensors in case of Gaussian sensor distribution and uniform hashing. GHT shows its usual unbalance problems, while DELiGHT manages to balance the load is



(a) GHT–`put` (b) DELiGHT–`put`

(c) GHT–`get` (d) DELiGHT–`get`

Fig. 6: Mean and standard deviation of the costs (number of hops) of `put` and `get` with Gaussian distribution.

able to balance the load (despite uniform hashing) because it keeps replica distribution localized and avoids replication on long perimeters (which happens with GHT in low density areas). This behavior is even more evident in Figure 5.(e,f) in which we compare the load of GHT and DELiGHT in case of Gaussian distribution of sensors and Gaussian hashing.

Evaluating `put`*s and* `get`*s.*

Figure 6 shows the mean and standard deviation of the cost of the basic `put` and `get` operations (number of hops needed to store a data). We performed 2000 `put`s and 2000 `get`s with randomly chosen meta-data. The QoS for DELiGHT is again pure replication with 15 replicas for each datum. In all graphs, the x axis shows the sensor density in the network and the y axis the operation cost measured. Graphs in the first row compare the cost of a `put` operation in GHT (a) and DELiGHT (b). The `put` is much more efficient in DELiGHT as it keeps the replicas localized in a ball without following long perimeters across the network. On the other hand, Figure 6.(c,d) shows the mean and standard deviation of the cost of a `get` operation with GHT (c) and DELiGHT (d). The cost of DELiGHT is greater than GHT. This is due to the fact that in GHT, as soon as a `get` request hits a node in the perimeter it immediately finds the data, on the other hand, using DELiGHT that request must travel until it reaches the replication ball, which may need more hops. In fact, if meta-data is hashed outside the external perimeter, a `get` operation could mean traverse the entire network before hitting the ball where the data was stored.

Note however, that the higher costs incurred in the `get` of DELiGHT is largely conterbalanced by the much larger cost of the `put` of GHT, and that this additional cost also provides load balance and QoS in the network. Note also that the `put` operations are expected to be much more common than `get`s during the network lifetime.

Density estimation evaluation.

In this section we show the results of the simulations that we performed to select an estimation function to approximate the sensor distribution.

Section 3 details the motivation to have an heuristic that estimates the density of the sensors.

The geographical points to place the *probes* are chosen as a regular pattern of k^2 points. We performed simulations with $k^2 = 9, 16, 25, 36, 49$.

Let $A_{x,y}$ be the area around the a geographical point (x, y) of which we want to estimate the density, and (x_i, y_i, d_i) the coordinates and the density measured by the *probes*. The estimation functions that we test are:

1. the mathematical mean of the densities measured by the *probes*
2. the theoretical density of the network
3. the density measured by the nearest *probe*
4. $\rho(x,y) = \frac{\sum_i \frac{d_i}{(x-x_i)^2+(y-y_i)^2}}{\sum_i \frac{1}{(x-x_i)^2+(y-y_i)^2}}$ with the summation performed on all the *probes* of the network
5. $\rho(x,y) = \frac{\sum_i^3 \frac{d_i}{(x-x_i)^2+(y-y_i)^2}}{\sum_i^3 \frac{1}{(x-x_i)^2+(y-y_i)^2}}$ with the summation performed on the 3 *probes* that are closer to (x, y).

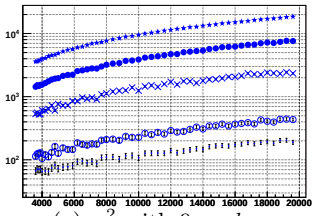
The difference between heuristics 4 and 5 is that the latter performs the weighted mean only on the 3 *probes* of the PD (`Probed Density`) that are closer to (x, y) while the former uses the entire PD.

The other simulation parameters are:

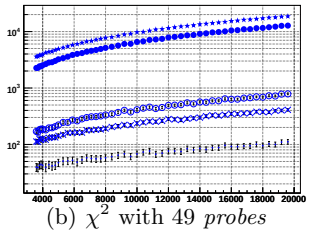
- network area is a square with a 400m side
- sensor transmission range is 10m
- the number of sensors is between 3600 and 20000
- sensor distribution can be uniform or gaussian

The performance of the estimation is measured using the χ^2 of the estimation with a number of degrees of freedom equals to the number of nodes minus the number of *probes*. The χ^2 is divided by the number of degrees of freedom. As shown in [14], the χ^2 function should approach 1.

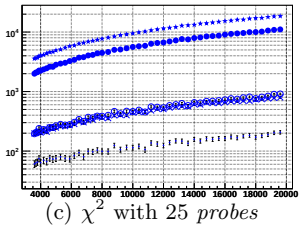
Note that the gaussian distribution is approximated using polinomial functions. The motivation is that we look for a general solution to the estimation problem. Thus, we do not exploit the knowledge of the real distribution function.



(a) χ^2 with 9 *probes*



(b) χ^2 with 49 *probes*



(c) χ^2 with 25 *probes*

Fig. 7: χ^2 for different Heuristics. From top to bottom, Heuristics 1,2,3,4 and 5.

The results of the simulations show that an uniform distribution of sensors is estimated equally well by every estimation function we proposed. More interesting are the results on gaussian distributed WSNs.

The first two heuristics have a lousy performance. In fact, they calculate a global estimation, independent from the geographical point the density is estimated on.

Heuristic 3 performs slightly better, but it does not implement a real interpolation of the densities measured by the *probes*.

Heuristic 4 does a poor job, caused by the non-locality of the data that are used in the estimation.

Heuristic 5 significantly outperforms the other heuristics. The χ^2 can not approach 1, because the estimation function we used is not a gaussian.

Figure 7.a shows a comparison of the different heuristics with 25 *probes*. Figure 7.b shows the same comparison with 49 *probes*. The behavior of the estimation functions are clearly visible, and it is seen that quality of the approximation with more than 25 *probes* does not provide significant improvements, while with less than 25 *probes* the quality is poor.

6 Conclusions and future work

In this paper, we discussed the limitations of the GHT in practical WSNs and we have proposed a new protocol (DELiGHT) which overcomes these limitations and allows a fine QoS control by the user. DELiGHT corrects the

ill behavior of GHT in three ways. (1) It uses non uniform hash tables, which allow a much balanced distribution of data across the WSN when sensors are distributed in a non uniform way. (2) The *dispersal protocol* used in DELiGHT allows the user to control the number of replicas of a given datum (using the quality of service parameters in the `put`). (3) Data replicas are placed in sensors which are "as close as possible" to the home node, which result in much balanced load on all the WSN. The merits of DELiGHT have been evaluated through simulation in uniform and Gaussian distributed WSNs. The results show that the protocol performs a better load balancing with respect to GHT, and has a smaller cost for `put` operations.

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. 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
3. 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
4. Yao, Y., Gehrke, J.: The Cougar Approach to In-Network Query Processing in Sensor Networks. SIGMOD Record **31**(3) (2002) 9–18
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. Bose, P., Morin, P., Stoimenovič, I., Urrutia, J.: Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. Wireless Networks, **7**(6) (2001) 609–616 Also in *DialM'99*, Seattle, August 1999, 48–55.
7. Albano, M., Chessa, S., Nidito, F., S.Pelagatti: Q-night: Adding QoS to data centric storage in non-uniform sensor networks. Technical report, Dipartimento di Informatica, Università di Pisa (2006)
8. Albano, M., Chessa, S., Nidito, F., Pelagatti, S.: Geographic Hash Tables with QoS in non Uniform Sensor Networks. ACM Mobihoc 2006 (Poster Session), Firenze, Italy, 22-35 May 2006, pp. 3.
9. Neumann, J.V.: Various techniques used in connection with random digits. In Taub, A.H., ed.: John von Neumann, Collected Works. Volume 5. Pergamon Press, Oxford (1951) 768–770
10. Jaromczyk, J., Toussaint, G.: Relative neighborhood graphs and their relatives. Proc. of IEEE **80**(9) (1992) 1502–1517
11. 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)
12. 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
13. 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
14. Wilson E.B., Hilferty M.M.: The distribution of chi-square In Proc. Nat. Acad. Sci., Wash. DC, 17, 684-688. (1931)