Università di Pisa

Dipartimento di Informatica

Technical Report: TR-06-16

Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks

M. Albano[ab], S. Chessa[ab], F. Nidito[a], S. Pelagatti[a]
[a] Dipartimento di Informatica, Università di Pisa
[b]ISTI-CNR Pisa
michele.albano@isti.cnr.it, {ste,nids,susanna}@di.unipi.it

October 25, 2006

Address: via F. Buonarroti 2, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks

M. Albano[ab], S. Chessa[ab], F. Nidito[a], S. Pelagatti[a]
[a] Dipartimento di Informatica, Università di Pisa
[b]ISTI-CNR Pisa
michele.albano@isti.cnr.it, {ste,nids,susanna}@di.unipi.it

October 25, 2006

Abstract

Storage of sensed data in wireless sensor networks is essential when the sink node is unavailable due to failure and/or disconnections, but it can also provide efficient access to sensed data to multiple sink nodes. Recent approaches to data storage rely on Geographic Hash Tables for efficient data storage and retrieval. These approaches however do not support different QoS levels for different classes of data as the programmer has no control on the level of redundancy of data (and thus on data dependability). Moreover, they result in a great unbalance in the storage usage in each sensor, even when sensors are uniformly distributed. This may cause serious data losses, waste energy and shorten the overall lifetime of the sensornet. In this paper, we propose a novel protocol, Q-NiGHT, which (1) provides a direct control on the level of QoS in the data dependability, and (2) uses a strategy similar to the rejection method to build a hash function which scatters data approximately with the same distribution as sensors. The benefits of Q-NiGHT are assessed through a detailed simulation experiment, also discussed in the paper. Results show its good performance on different sensors distributions on terms of both protocol costs and load balance between sensors.

1 Introduction

A wireless sensor network is composed by a large number of low power, low cost sensors (also called nodes) [1] which self organize into a (multi-hop) ad hoc network. A sensor is a micro-system which also comprises one or more sensing units, a radio transceiver and an embedded battery. Sensors are spread in an environment (the *sensor field*) without any predetermined infrastructure and cooperate to execute common monitoring tasks which usually consist in sensing environmental data from the surrounding environment. The sensed data are collected by an external sink node when it is available (connected to the network). The sink node, which could be either static or mobile, is in turn accessed by the external operators to retrieve the information gathered by the network.

Data in a sensornet can be accessed according to different paradigms which define the communication model between sensors and between sensors and sink node. In principle, sensed data can flow to the sink as soon as they are sensed. However, this causes a large amount of unneeded communications to take place and a large amount of power to be wasted in delivering row series of sensed data. More recent paradigms [2, 3, 4], see the sensornet as a distributed database. The sensornet is then 'programmed' by the the sink node by sending queries to sensors. Sensors in turn reduce and filter the sensed data locally before sending an answer to the query.

Data Centric Storage. As the sensor network scales in size, so does the amount of sensed data which is processed and collected by the network. In the effort to provide efficient access to data and to tolerate disconnections between the network and the sink node, recently the use of Geographic Hash Tables (GHT) has been proposed to implement a Data Centric Storage (DCS) within the network [5]. DCS defines an innetwork data storage technique that locates data on the basis of data name rather than node addresses. This is appropriate for sensornets as the identity of the individual node that gathered the data is usually not relevant. DCS applies to event detection applications in which sensed data are stored in the network for later user retrieval. Retrieval requests can be formulated via the data name (which is enough to identify the data location) and efficiently performed via a unicast request message. DCS can represent the analogous of the *storage management layer* in a DBMS, as it takes care of locating and storing data around the network. This can be the basis for building more high level DB abstractions (as in TinyDB[3]).

Geographic Hash Table (GHT). GHT [5] is a DCS implementation using hash functions to distribute data uniformly across the network. Each datum is associated a unique name (or *metadatum*).

When new data are stored, the corresponding name is hashed into a location (x, y) in the (two-dimensional) sensor field. Then, GHT uses GFG [6][*] to select the group of sensors surrounding (x, y), called the *perimeter*. A datum corresponding to (x, y) is stored in all sensors belonging to the perimeter of (x, y). Storing on all the perimeter is essential to guarantee data persistence also in presence of node faults.

Data retrieval is also performed using GFG. The name is first hashed into the destination (x, y), then GFG is used to route the request (x, y). Sooner or later the request will hit a node in the perimeter of (x, y), which will send back the requested data.

[*]In [5] the authors call this protocol GPSR [7].

Load unbalance and QoS in GHT. In the first part of this paper we analyze the characteristics of GHT simulating its behaviour with uniform and Gaussian distributions of sensors. Results show that the amount of data distributed to each sensor can have a large variance, causing hot spots on borders of the data field even with uniform sensor distribution. This may cause data losses due to lack of storage in the boarder nodes and to battery exhaustion. Moreover, dead sensors on the borders may cause further unbalance due to ill shaped perimeters. As we may expect, the effect of load unbalance is more serious when sensors distribution is not uniform. This is due to the fact that names are hashed by GHT uniformly on the sensornet. This phenomenon becomes worse in the non uniform case due to the fact that the GHT hash function distributes keys on the field regardless of the actual sensor density.

In particular, we simulated the behavior of GHT with both uniform and non uniform (i.e. Gaussian) distributions of the nodes. Results show clearly that the amount of data distributed to each sensor can have a large variance, causing hot spots on borders of the data field.

Moreover, GHT (as well as other DCS methods proposed so far, such as CHR [8]) does not give control to the programmer over the *quality of service* (QoS) deserved for a given datum. Generally speaking, we can define the QoS as the dependability required by the data, which in turn may be expressed using different metrics and ranges according to the particular redundancy technique used. For instance, using pure replication we may have different levels of QoS depending on the number of actual replicas ensured for a given datum. Unfortunately, in GHT number of replicas of the datum depends on the perimeter of its hash (x, y) and cannot be bound to the importance of the datum to be stored. Incorporating QoS control in GHT requires a new protocol in which data distribution has a direct control on the service given to each datum.

Q-NiGHT–Quality of service in Non uniform Geographic Hash Tables. In this paper we propose Q-NiGHT, a novel DCS protocol which moves from GHT incorporating QoS control and featuring good load balance among sensors. Q-NiGHT uses a strategy similar to the *rejection method* [9] to build a hash function biased with sensor distribution. This is the basis to spread data more evenly among nodes. In addition, Q-NiGHT 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 a detailed simulation of Q-NiGHT 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 Q-NiGHT on different sensors distributions on terms of both protocol costs and load balance.

The paper is organized as follows. Sec. 1.1 introduces some definitions and symbols used in the paper. Sec. 2 discusses the problems of GHT on sensor networks presenting the results of some simulations carried on with uniform and Gaussian distributions. Sec. 3 discusses the 'biased' hash function h used in our protocol. This function scatters keys according to a given sensor distribution and is used instead of the usual uniform hashing functions. Then, Sec. 4 details Q-NiGHT and discusses how different redundancy techniques can be incorporated in the protocol. Finally, Sec. 5 reports on the simulation of our protocol and compares its performance with plain GHT. We present some related work in Sec. 6 and our conclusions in Sec. 7.

1.1 Notation

Recurrent symbols used in the paper are summarized in Tab. 1.

$s\ s_i\ s_j$	generic sensors
n	number of sensors in the network
r	communication range of a sensor
A	deployment area in which sensors are located
f	geographical distribution of sensors in A
h	hash function used to locate data
D	datum to be stored/retrieved, in the system
M	name (meta-data) for D

Table 1: Recurrent symbols

We also planarize the input graphs using both RNG (*Relative Neighborhood Graph*) and GG (*Gabriel Graph*), defined as follows [10].

Let $S = \{s_1, s_2, ..., s_k\}$ be a set of nodes in $\mathbb{R}^2$ and $\delta_{i,j}$ the distance between s_i and s_j in S.

Definition 1 (RNG) *The RNG of S is a graph with nodes in S and an edge between s_i and s_j if and only if there is no node $s_k \in S$ such that*

$$\max\{\delta_{i,k},\ \delta_{k,j}\} \ \leq\ \delta_{i,j}$$

Definition 2 (GG) *The GG of S is the graph with nodes in S and an edge between s_i and s_j if and only if there is no node s_k such that $\delta^2_{i,k} + \delta^2_{k,j} \ \leq\ \delta^2_{i,j}$.*

Finally, we define *node density* of the graph as the mean number of neighbors per node.

2 Load Unbalance in GHT

GHT [5] implements data Centric Storage using Geographic Hash Tables. Each datum has a unique metadatum (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.

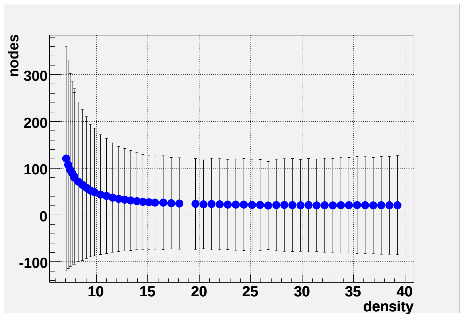
In the `put` operation, the name of data to be stored is first hashed into a location (x, y) in the sensing field. Then, GHT selects the sensor closest to (x, y), which becomes the *home node* for that data. The home node is selected using GFG. GFG uses two operation modes: Greedy and perimeter. Each 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*, that is 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 [5].

Data retrieval uses a `get` operation. The name is first hashed into the destination (x, y), then GFG is used to route the request (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 rather 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, in practice, sensing fields are bounded and/or have got non-uniform sensor distributions, thus we can expect a heavily unbalanced load, as the length of the perimeters found by GFG exhibits high variability.

In order to measure the degree of unbalance of GHT on more realistic scenarios, we set up the following experiment. We simulated a flat square sensing field, with a 400 meters side. We assumed the area populated by identical sensors, with a circular transmission range with 10 meters radius. In this area, we simulated several sensornets ranging from 3600 to 20000 sensors, which correspond to a mean network density ranging in $[8, 40]$. For each density in $[8, 40]$, we randomly generated 100 sensornets with uniform distribution. Then, for each sensornet, we compute the mean/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, in the case that GG or RNG are used in the protocol. Then, we compute mean and variance on the perimeters measured. Fig. 1, summarizes the mean and standard deviation of the perimeters for different densities of the sensornet (from 8 to 40) and different planarization (GG and RNG). Form the figure, we can see that, as the network density increases, the average number of nodes in a perimeter decreases. However, the actual number of nodes remains highly variable, and the standard deviation is higher with RNG than with GG. This variance is partly due to the behaviour of nodes in the outer part of the sensing area (the grey part in Fig. 2), since in that area



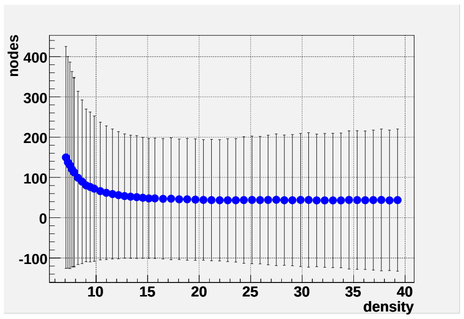


Figure 1: Mean and variance of perimeters (number of nodes) measured for different sensornet densities in case of GG (left) and RNG (right) planarization.

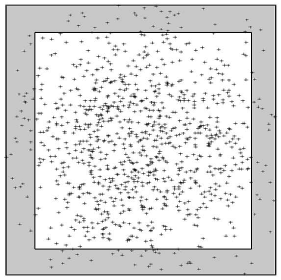
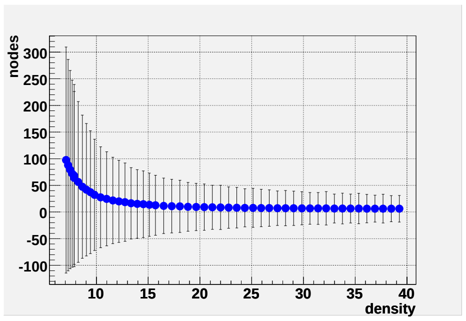



Figure 2: *(left)* The border area (gray) and the storing area (white). *(right)* Mean and variance of perimeters measured for different sensornet densities in case of GG using only the white area.

the probability of having very long perimeters (ie following the whole boarder) is high. In particular 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 quite hight.

Perimeters on the border. In order to understand this *border effect* better, we performed another set of simulations in which the sensing nets are generated in the same way as above but the boarder area is not used to store data. In these simulations, we "cut away" the 5% of the area from each border (the grey are in Fig. 2.*left*) for a total of 19% of the total area. Then we randomly generate 1000 points in the white area and again measure the length of each perimeter and compute mean and variance. In Fig. 2.*right*, we show the results obtained using GG. As we can see from the picture, the mean and variance improve if the border nodes are left out but standard deviation remains high, leading to a high unbalance in the node load. Results with RNG are much worse, as happened with the experiment considering the whole sensing area (Fig. 1).

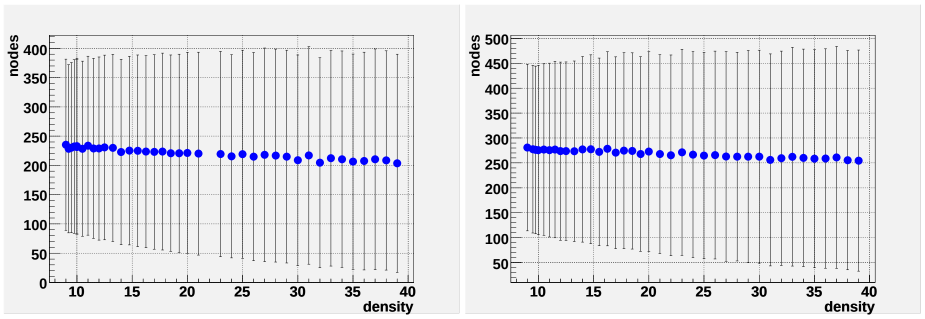


Figure 3: Mean and variance of GHT perimeters for different sensornets densities, Gaussian sensor distribution, with GG (left) and RNG (right), using all the sensing area.

Non uniform sensor distribution. In order to understand the behaviour 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 our sensors. The function is normalized to have the 99 percentile matching the area. The results are shown in Fig. 3. Here, the behaviour is much worse than with uniform distribution. Unfortunately GHT, uses a uniform hash function independently of the real distribution of sensors. This bring to a pathological state of *load unbalance*. The load unbalance 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. A last problem 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 of the actual meaning of the data. 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 [5] introduce the technique of *structured replication*. However nor GHT, neither the structured replication ensure that the level of redundancy associated to a data is related to the importance of the data itself. In practice, GHT assure only the same average treatment of each stored data.

A second important 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.

This means that GHT and structured replication as proposed in [5] are

```
RejectionHash(k, f): <x,y> {
   i = 0;
   while(true)
     let <x,y,z> uniformly hashed
          from k+i in box(f)
     i = i+1
     if z < f(x,y) return <x,y>
   }
```

Figure 4: Pseudocode for the rejection method

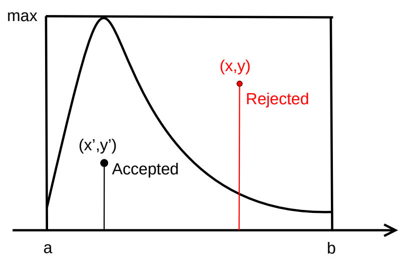


Figure 5: Rejection method: A point is accepted only if it is under the curve

unsuitable to offer a given QoS, since they cannot guarantee a given level of redundancy to a data which is possibly related to its importance.

3 Nonuniform Hashing

As we have seen, a serious problem with GHT is due to the fact it uses a uniform hash function independently of the real distribution of sensors. This lead to the pathological load unbalancing shown in Fig. 3. Our solution to this problem is to use hash functions which scatter data approximately with the same distribution as the sensors. To do this, we first observe that 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 in a value in $\mathbb{R}^2$) such that for *near* values of key the hashed values must be *distant*. With this consideration in mind we define a new hash function, whose pseudo-code is shown in Fig. 4. This function uses a strategy similar to the one used in 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 shown in Fig. 5. 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 values are below the function. Notice that in principle there is a non-null probability of non termination because the values can be generated all above the function. In practice a good uniform hash grants to generate values in all the box. We use the rejection method because $h(k)$ must work with any distribution. Rejection method has some problem in the generation of values belonging to functions that have *long tail* or that are very steep. These two kinds of distributions are not present in sensor networks. A network is always limited in space and steep distributions are difficult to obtain with random deployment. In Fig. 4, function `RejectionHash` returns a pair (x, y) of coordinates where to place data from its key `k`, belonging to distribution `f`. The function simply uses a standard uniform

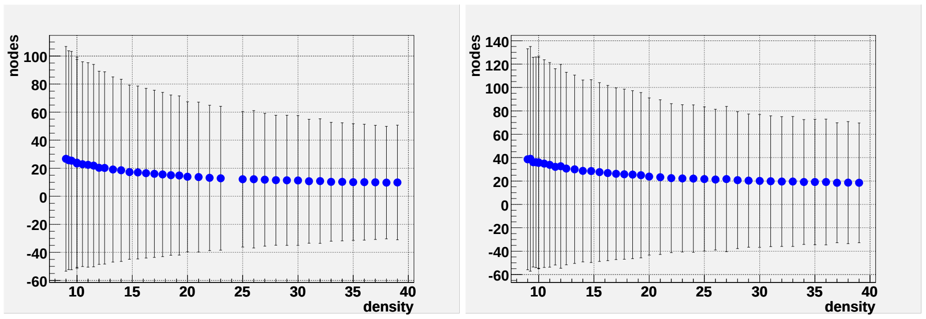


Figure 6: Non uniform hashing. Mean and variance of GHT perimeters for different sensornets densities, Gaussian sensor distribution, with GG (left) and RNG (right), using all the sensing area.

hash function to generate a triple (x, y, z) in the box around `f`, If the point generated is below the input function `f`, the pair (x, y) is returned as a valid coordinate, else it changes the key (in the algorithm we use the sum operation to represent this) and hashes it again until the return condition is satisfied.

Function `RejectionHash` is a simple extension of rejection method for the generation of points in $\mathbb{R}^2$. The method is mathematically correct [9] for the $\mathbb{R}^1$ case. *Correct* means that values produced by the rejection method always belong to the original function `f`. The correctness in $\mathbb{R}^d$ is proven in Appendix B. Fig. 6 shows a good behavior of the non-uniform hash function. `RejectionHash` fits well the the sensor distribution 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 (Fig. 1 and Fig. 2 right). 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 Q-NiGHT a New GHT Protocol with QoS

As well as GHT [5], Q-NiGHT is built atop the GFG routing protocol[6]. Q-NiGHT provides data insertion (via `put`) and data retrieval (via `get`) on the sensor network.

To our purposes the interface of the `put` includes, along with the meta-data M and the data D, also a parameter Q expressing the desired QoS. Q gives a measure of the dependability required for the data, may be expressed using different metrics and ranges according to the particular redundancy technique used.

For instance, if Q-NiGHT adopts pure replication then Q can express the number of sensors on which the data should be replicated, or, if Q-NiGHT adopts n out of m redundant encodings (such as [11, 12]), then Q can express the number of fragments in which partition the data (each fragment to be stored in a different sensor) and the number of redundant fragments. In the following, we first describe Q-NiGHT assuming pure replication of the data. Extensions to different replication techniques is discussed afterwards. Data insertion is involved with `put(M,D,Q)`. We assume Q ranges in $[1, Q_{max}]$ and gives the number of sensors on which the data should be replicated. Let s be the source node of a `put(M,D,Q)` operation. s firstly computes $h(M)$, where h is the hash function conditioned with the sensor distribution function, f, in the sensing field, as discussed in Sec. 3. $h(M)$ returns a pair of geographic coordinates (x, y) as the destination of the packet $\mathtt{P}_p$`=<(x,y),<M,D,Q>>`. The packet in turn is sent to the destination using the GFG 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.[†] 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 $B_{(x',y')}(2\overline{r}) - B_{(x',y')}(\overline{r})$ (ie, the ones not already reached by the first iteration). The dispersal protocol stops as soon as Q sensors have been hired or the outermost perimeter has been reached. Our dispersal protocol is really simple. It can be seen as a geocasting protocol[13] in which we use a runtime computation of the size of the delivery region.

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 $B_{(x',y')}(\overline{r})$.[‡] In simulations, to compute $\overline{r}$, we use a simple strategy based on the approximation of the

[†]This property is proved formally in Appendix A.

[‡]An approximation strategy for $\overline{r}$ (given Q and f) is discussed in Appendix A.

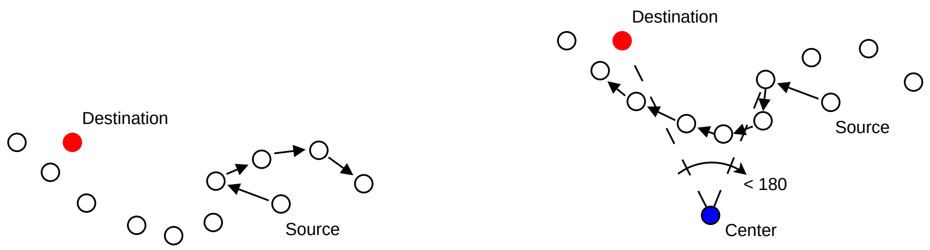


Figure 7: GFG routing perimeter mode (left) and our enhanced GFG strategy (right)

ball with its inscribed square. Using this approximation, we are sure that the method converges in 2 iterations in almost the totality of the cases and the average number of messages generated is minimal.

Enhanced GFG. Our protocol actually uses a slightly modified version of GFG, which appears to be more efficient with non uniform sensor distributions. Usually, when GFG is in the *perimeter mode*, it always adopts clockwise turn to reach the destination coordinates. This behavior leads to pathological situations as the one shown in Fig. 7.*left*. Here, the perimeter is very long, and while the source and the destination are really close, the hand side rule make the packet traverse all the perimeter before reaching the destination node. This is not a problem for GHT as the data are replicated on all the perimeter, but may be very inefficient for Q-NiGHT, which replicates only on a ball surrounding the destination. In our GFG version, we turn clockwise or counterclockwise depending on the destination, as shown in Fig. 7.*right*. Let s_a be the position of the sender node, c the position of the center of the deployment area and s_d be the position of the destination. We turn clockwise if $0 < \widehat{s_a c s_d} < 180$[§], and counterclockwise otherwise.

Behavior in case of faults. In case some of the nodes holding the replicas of `<M,D>` break down our protocol continues to operate correctly. Due to the characteristics of GFG, any `get` with key `M` is routed to the node geographically nearest to $(x, y) = h(M)$, that is the home node for `M`. This implies that if the faulty node is not the home node, the protocol implicitly discards it.

If this is not the case, we can prove that the second nearest node in the perimeter is always included in the ball built by our protocol.

We proof this property as follows. Let (x', y') be the home node for data located in (x, y) and let $\overline{r}$ be the radius computed as in Appendix A in such a way that $B_{(x,y)}(\overline{r}) = Q$ with high probability (w.h.p.). Let us set $\overline{t} = \overline{r} + \delta$, where δ is the Euclidean distance between (x, y) and (x', y'). We grant that irradiating the data from (x', y') using a ball of radius $\overline{t}$, we reach all the Q nodes in $B_{(x,y)}(\overline{r}) = Q$ w.h.p..

[§]computed in clockwise way

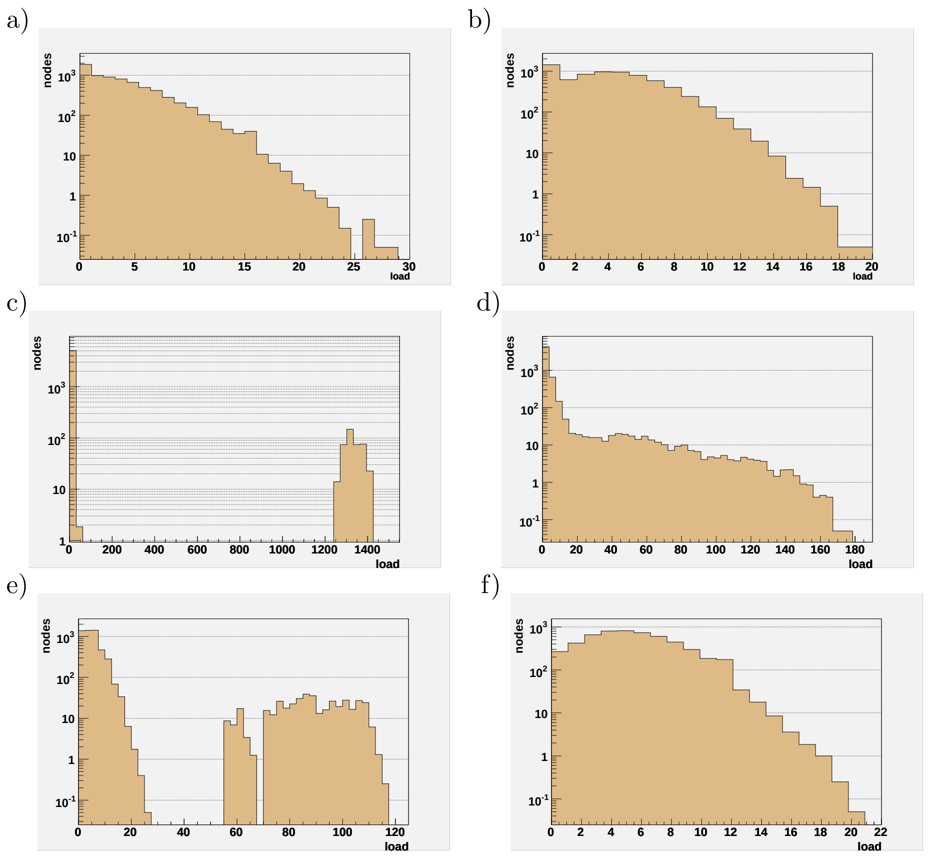


Figure 8: Average load of sensors: Uniform sensor distr. and uniform hashing and RNG (*a* GHT, *b* Q-NiGHT), Gaussian sensor distr. and uniform hashing (*c* GHT, *d* Q-NiGHT), Gaussian sensor distr. and Gaussian hashing (*e* GHT, *f* Q-NiGHT)

The correction $\overline{t} = \overline{r} + \delta$, assures that in $B_{(x,y)}(\overline{t})$, that is irradiated by (x', y') there are all the nodes that were in $B_{(x,y)}(\overline{r})$ because the second ball is completely included in the first one. Then the protocol, builds the set of sensors that home data with the Q nodes nearest to (x, y), thus all the nodes that were in $B_{(x,y)}(\overline{r})$.

5 Simulations and Results

In this section, we discuss the results of our simulation. We simulated a square with a 400 meters 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 metadata using both GHT and Q-NiGHT . In these trials,

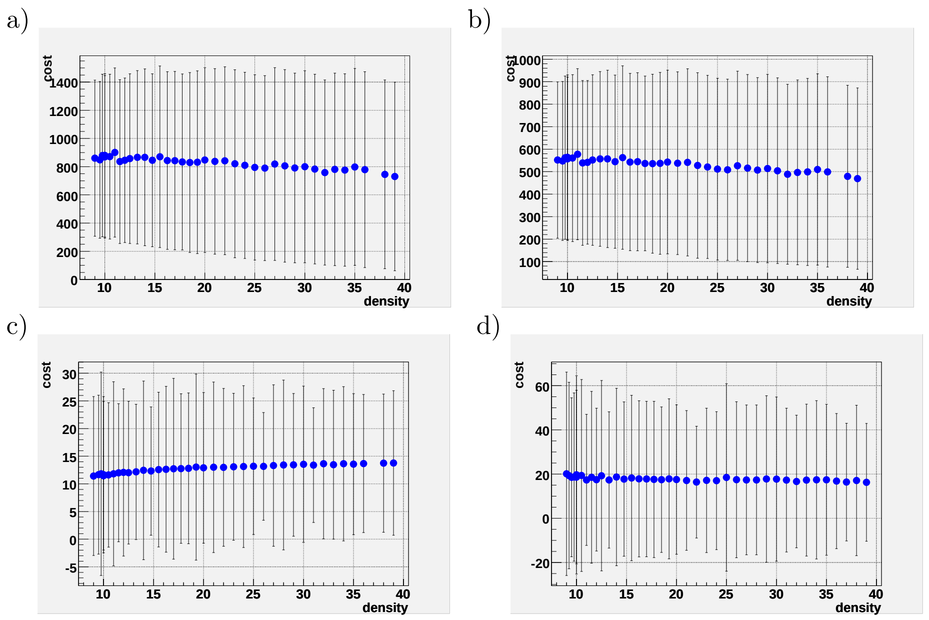


Figure 9: Costs (number of hops) of `put` and `get` with Gaussian distribution of sensors and RNG. (a, b): Mean and standard deviation of `put` cost (GHT (a), Q-NiGHT (b)). (c,d) Mean and standard deviation of `get` cost (GHT (a), Q-NiGHT (b)). Here Q-NiGHT uses the enhanced GFG.

Q-NiGHT uses a pure replication QoS with 15 replicas for each datum.

Fig. 8 compares the behaviour of GHT (graphics a, c, e on the left) and Q-NiGHT (graphics b, d, f on the right). All graphics show the average load of sensors using RNG. Graphs (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 see that Q-NiGHT reaches better load balance even in the uniform case (graphs a, b). In Fig. 8.(c, d), we compare the average load of sensors in case of Gaussian sensor distribution and uniform hashing. Here, GHT shows its usual umbalance problems, while Q-NiGHT manages to balance the load is 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 Fig. 8.(e, f) in which we compare the load of GHT and Q-NiGHT in case of Gaussian distribution of sensors and Gaussian hashing.

Evaluating `puts` and `gets`. Fig. 9 shows the mean and standard deviation of the cost of the basic `put` and `get` operations (number of hops needed to store a datum). Here, we performed 2000 `puts` and 2000 `gets` with randomly choosen metadata. The QoS for Q-NiGHT is again pure replication with 15 replicas for each datum. We always consider RNG networks. In all graphs, the x axis shows the sensor density in the network and the y axis the operatin cost measured.

Graphs in the first row compare the cost of a `put` operation in GHT (a) and Q-NiGHT (b). The `put` is much more efficient in Q-NiGHT as it keeps the replicas localized in a ball without following long perimeters across the network. On the other hand, Fig. 9.(c,d) shows the mean and standard deviation of the cost of a `get` operation in case of RNG networks with GHT (c) and Q-NiGHT (d). Here, the cost of Q-NiGHT 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 Q-NiGHT the request must travel until it reaches the replication ball, which may need a few more hops. This behaviour was much worse using standard (not enhanced) GFG as travelling along the perimeter could mean traverse the entire network before hitting the ball. However, in our opinion the cost of `get` is still rather low as a price to be paied in order to get load balance on the network and QoS. Moreover, we can expect that the `put` operations wuold be much more common during the network lifetime.

6 Related Works

Data Centric Storage. Our work obviously originates from the GHT [5] which has been already described in much detail in the rest of the paper. With respect to this work, Q-NiGHT improves the behavior of Geographic Hash Tables in three important aspects. First, it uses nonuniform hash tables, which allow a much balanced distribution of data across the sensor network when sensors are distributed in a non uniform way. Second, the *dispersal protocol* used in Q-NiGHT allows the user to control the number of replicas of a given datum (using the quality of service parameters in the `put`). Third, 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 sensor network.

Cell Hash Routing (CHR)[8] is another DCS routing based on hash tables. CHR first clusters nodes in cells of predefined and globally known shape using a distributed protocol (e.g. dividing the sensor field in a mesh of squares). Then it uses the cells, instead of individual nodes to hold the values. CRT uses a variant of GFG working on cells. Data are hashed into geographic coordinates as in GHT[5] and routed to the cell that includes that location. As soon as the data reaches the destination cell it is stored in all the nodes in the cell. If the cell is empty (e.g. it has no sensor) CHR uses an approach similar to GHT to replicate data *around* the empty cell.

Problems related with using geographic face routing in practical settings have been discussed in [14]. To solve this problems new solutions were proposed.

For instance, GEM [15] proposes a graph embedding for sensor networks which defines a set of virtual coordinates that can be used for routing and DCS in place of true geographic coordinates. In particular the authors propose to use their coordinate system and routing algorithm instead of GFG for data dissemination. This and other proposal based on virtual coordinate system are orthogonal to ours because our protocol can be adapted to use a routing algorithm different from GFG, provided that we find a sound equivalent to our sensor ball defined using the virtual system.

Another approach similar to GEM is Hierarchical Location Identifiers (HLI) [16]. The system uses a hierarchical location sistem to identify nodes by some characteristics, the authors state that this system is more convenient than geographic position. The system uses DSDV[17] routing protocol and aggregate routes. This provides a good base to perform unicast, multicast and anycast. The system, authors claim, can be used as a base for other solutions as tinyDB and GHT.

Information Directed Routing. The problem of DCS is related also to anoter problem known as Information Directed Routing (IDR)[18]. To route a message, IDR finds *a path with maximum information gain at moderate communication cost.* A query message is sent from a source node to a destination node. The network routes the message from the souce to the destination finding a path that is not minimal in terms of energy but this path pass through *high interest areas* to collect as much information as possible to reply to the query. This approach is an optimization of Directed Diffusion[2] protocol. The message is not broadcasted to every one. Instead it is routed only to interest regions to collect data.

Non uniform hashing. The problem of non-uniformity in relation with data storage and dissemination as been analyzed in various papers but with different meanings if related to our work.

In [19] the non-uniformity is related to the concept of *non-uniform information granularity.* Non-uniform information granularity means that the required accuracy, or precision, of information is proportional to the distance between the producer and the consumer of the information.

In [20] the non-uniformity is related to the sampling of data from a sensor network with *approximately uniform* methods. Authors pick data form the network using a rejection strategy using an uniform sampling on the sensors space. Sensor space is normalized to an uniform space using Voronoi regions computed in a distributed fashion.

7 Conclusions and Future Works

In this paper, we have discussed the limitations of the GHT in practical sensor networks and we have proposed a new protocol (Q-NiGHT) which overcomes these limitations and allows a fine QoS control by the user. The merits of

Q-NiGHT have been evaluated through simulation in uniform and Gaussian distributed sensor networks. The results show that the protocol performs a better load balancing than GHT, and has a smaller cost for `put` operations.

We also proposed an enhanced version of GFG protocol to correct the ill behavior of perimeter mode in case of long perimeters always explored clockwise. Future work will include the study of protocols to estimate the sensor distribution on the deployment area. Moreover, we would work on new routing protocols that work better in non-uniformly distriuted networks. We will also explore the use of the modified GFG protocol proposed in [21] and the opportunity to use virtual coordinates systems [22] instead of GPS-based positioning systems.

Acknowledgments

This work has been partly by the European Commission in the framework of the "SatNEx II" NoE project (contract N. 27393).

References

[1] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine **40**(8) (2002) 102–114

[2] Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In: Proc. of MobiCom 2000, Boston, MA (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, CA (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., Stoimenoviȁ, 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] Karp, B., Kung, H.T.: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In: Proc. of MobiCom 2000, Boston, MA (2000) 243–254

[8] 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)

[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. Proceedings of IEEE **80**(9) (1992) 1502–1517

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

[12] Rizzo, L.: Effective erasure codes for reliable computer communication protocols. ACM Computer Communication Review **27**(2) (1997) 24–36

[13] Seada, K., Helmy, A.: Efficient and robust geocasting protocols for sensor networks. Computer Communications **29**(2) (2006) 151–161

[14] Kim, Y., Govindan, R., Karp, B., Shenker, S.: On the pitfalls of geographic face routing. In: Proc. of ACM/SIGMOBILE DIAL-M-POMC 2005, Köln, Germany. (2005)

[15] 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, CA (2003) 76–88

[16] 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, NY, ACM Press (2004) 305–306

[17] Perkins, C.E., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In: SIGCOMM '94, New York, NY, ACM Press (1994) 234–244

[18] Liu, J., Zhao, F., Petrovic, D.: Information-directed routing in ad hoc sensor networks. In: WSNA '03: Proceedings of the 2nd ACM Int. Conf. on Wireless Sensor Setworks and Applications, New York, NY, ACM Press (2003) 88–97

[19] Tilak, S., Murphy, A., Heinzelman, W.: Non-uniform information dissemination for sensor networks. Proc. of the 11th International Conference on Network Protocols (ICNP03), Atlanta, GA (2003)

[20] Bash, B., Byers, J., Considine, J.: Approximately uniform random sampling in sensor networks. Technical Report BUCS-TR-2004-031, Boston University Department of Computer Science (2004)

[21] Arad, N., Shavitt, Y.: Minimizing recovery state in geographic ad-hoc routing. In: Proc. of MobiHoc 2006, Florence, Italy (2006) 13–24

[22] 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)

A QoS Radius Computation

In this section, we show how to approximate the ball radius in the dispersal protocol to converge (ie, reach Q sensors w.h.p.) in at most 2 iterations in the majority of the cases.

More formally, let f be the sensor probability distribution function, Q an integer in $[1, Q_{max}]$, and (x, y) a point in $\mathbb{R}^2$. We want to fix r in such a way that $B_{(x,y)}(\overline{r}) \geq Q$ with high probability.

We first discuss the intuition behind our approximation schema. Let A_r denote the circle of radius r, we want to fix $\overline{r}$ such that

$$E[number\ of\ sensors\ \in\ A_{\overline{r}}] = Q. \tag{1}$$

Equation 1 can be rewritten in terms of the probability f:

$$n \cdot \int_{A_{\overline{r}}} f(x, y) \mathrm{d}A_{\overline{r}} = Q \tag{2}$$

where n is the total number of sensors in the network and the integral represents the probability to have a sensor in $A_{\overline{r}}$. Direct computation of the integral above is likely to need a large number of floating point operations in practical distributions, which is too energy demanding in our setting.

To simplify the computation of $\overline{r}$, we use the following strategy. Instead of using $A_{\overline{r}}$ we use the square inscribed in the circle of radius $\overline{r}$. This square has edge $\overline{r}\sqrt{2}$ and area $A'_{\overline{r}} = 2\overline{r}^2$. If we impose to have Q nodes in $A'_{\overline{r}}$, we grant at least Q nodes in $A_{\overline{r}}$. Equation 2 becomes:

$$n \cdot \int_{A'_{\overline{r}}} f(x, y) \mathrm{d}A'_{\overline{r}} = Q. \tag{3}$$

The volume identified by the integral can be represented in terms of r as follows

$$n \cdot 2 \cdot \overline{r}^2 \cdot h = Q \tag{4}$$

where h is the ideal height of the cylinder which volume is equivalent to the one of integral.

We use the height h to simplify the computation of $\overline{r}$: Sensors do not compute the integral but have a stepped version of the nodes distribution function f. A sensor chooses h as the minimum of heights, of steps involved by A', that is larger than 0. This means that the value of $\overline{r}$ identifies a number of sensors greater than Q. Now equation 4 can be used to find the minimum $\overline{r}$ that identifies a number of sensors greater than Q:

$$\overline{r} = \sqrt{\frac{Q}{2 \cdot n \cdot h}}. \tag{5}$$

Eq. 5 is the one used in the actual computation.

B `RejectionHash` Correctness

`RejectionHash` generates hash values belonging to some function f using a strategy similar to the rejection method [9]. It uses uniform hash to support this procedure. Uniform hash, as stated in Section 3, is similar to a pseudo-random number generator. We define a rejection method *correct* if the values generated by the hash function are distributed with the same probability of the values produced by function f.

To prove correctness of `RejectionHash` we prove the correctness of rejection method in $\mathbb{R}^d$.

Let f be the probability density function that we want to use as `RejectionHash` parameter. Let f be defined in $D \in \mathbb{R}^d$. Let g be a density on D from which we generate samples such that for $x \in D$

$$f(x) \leq cg(x).$$

Rejection method acts as follows

1. generate X from $g(\cdot)$;
2. generate U from $[0,1]^d$ uniformly;
3. if $U \leq \frac{f(X)}{cg(X)}$, return X; otherwise, go to 1.

The proof of the validity of the rejection method is the following.
Let Y be a generated random value. For all $A \subset D$

$$\Pr\left[U \leq \frac{f(X)}{cg(X)}\right] = \int_D \frac{f(x)}{cg(x)} g(x)\mathrm{d}x = \frac{1}{c}.$$

$$\begin{aligned}
\Pr[Y \in A] &= \Pr\left[X \in A | U \leq \frac{f(x)}{cg(x)}\right] \\
&= \frac{\Pr\left[X \in A, U \leq \frac{f(x)}{cg(x)}\right]}{\Pr\left[U \leq \frac{f(x)}{cg(x)}\right]} \\
&= c\Pr\left[X \in A, U \leq \frac{f(x)}{cg(x)}\right] \\
&= c\int_A \frac{f(x)}{cg(x)} g(x)\mathrm{d}x \\
&= \int_A f(x)\mathrm{d}x.
\end{aligned}$$

This implies that f is the density of Y.