Dynamic Random Replication for Data Centric Storage

Ángel Cuevas
Universidad Carlos III de Madrid
acrumin@it.uc3m.es

Manuel Urueña
Universidad Carlos III de Madrid
muruenya@it.uc3m.es

Gustavo de Veciana
University of Texas at Austin
gustavo@ece.utexas.edu

ABSTRACT

This paper presents a novel framework for Data Centric Storage in a wireless sensor and actor network that enables the use of a *randomly*-selected set of data replication nodes which also *change over the time*. This allows reducing the average network traffic and energy consumption by *adapting* the number of replicas to applications' traffic, while balancing energy burdens by varying their location. To that end we propose and validate a simple model to determine the optimal number of replicas, in terms of minimizing average traffic/energy consumption, from the measured applications' production/consumption traffic. Simple protocols/mechanisms are proposed to decide when the current set of replication nodes should be changed, to enable new applications and sensor nodes to efficiently bootstrap into a working sensor network, to recover from failing nodes, and to adapt to changing conditions. Extensive simulations demonstrate that our approach can extend a sensor network's lifetime by at least a 60%, and up to a factor of 10x depending on the lifetime criterion being considered.

Keywords

Wireless Sensor and Actor Network (WSAN), Data-Centric Storage (DCS), Random Replication, Epoch, Optimization.

1. INTRODUCTION

In this paper we consider a simple framework to build a distributed information delivery service for one or more applications running over a Wireless Sensor and Actor Network (WSAN). Each application is modeled as a (randomly) distributed set of *producer* and *consumer* nodes, e.g., sensors or actuators that can exchange information by relaying packets across neighboring nodes. We assume that producer and consumer nodes do not have explicit knowledge of each other, but are aware of the name(s) of the application(s) in which they are participating. This makes possible to build a highly scalable distributed information service involving large numbers of producers/consumers.

Data-Centric Storage (DCS) [1, 2, 3] is an elegant solution to this problem. The key idea is to identify one node in the network which serves as a rendezvous point between producers and consumers associated with an application. The node is determined by generating a spatial location based on applying a hash function to the application's name, and then finding the node in the network which is closest to it. Thus producers and consumers, which have knowledge of the hash function and application's name, are able to determine and route to a common rendezvous point without any additional information. A producer pushes new information to the rendezvous node, which, in turn, is responsible for storing (and possibly aging) data. Consumers are able to subsequently pull information from the same rendezvous point.

In this paper we consider a Data-Centric Storage framework where application's information is pushed, stored and/or replicated across a *set* of rendezvous points. This permits consumers to pull information from rendezvous points that are closer, possibly reducing traffic, energy overheads and reducing response time, while also improving fault-tolerance in the case where nodes fail or run out of energy. Additionally, in order to balance energy expenditures over time, we study an approach to vary the set of replication nodes over time. Specifically, we consider the case where nodes can determine the current set of N_r replicas associated with a given application by generating N_r random spatial locations with a hash function $hash(app \oplus epoch \oplus i) \ \ \forall i \in [0, N_r - 1]$, where *app* is the application's name and *epoch* is a shared time identifier employed to change replicas over the time. Network nodes that are the closest ones to these hashed spatial locations serve as rendezvous (or replication) nodes for the application. In this setting a producer and/or consumer, which is aware of the application's name, the current time epoch and N_r, can independently determine the location of the 'nearest' replication node by determining the minimum distance between itself and all spatial locations generated by the hash function.

As mentioned earlier, closeness between consumers and replication nodes is beneficial from the point of view of reducing traffic to consumers, energy expenditures and delay to access the data. However, if a large number of replication nodes is employed, the production costs, including the cost to transport and store information across multiple rendezvous nodes can be high. Thus a key trade-off in our framework is to decide how many rendezvous nodes should be used. For the case where the hash function results in roughly random spatial locations, we show precisely how this tradeoff can, and should, be optimized so as to mini-

mize the total network traffic, in bits-m/sec, and thus, to first order, also minimize the overall energy consumption of a given application. The optimal number of rendezvous nodes depends on the ratio of the production intensity to that of consumption, i.e., is critically dependent on the traffic associated with the application.

In the case where the consumption intensity dominates production, data is replicated across all replication nodes, whereas in the opposite case producers store the data *solely* at the closest rendezvous node, and so consumers query all the rendezvous nodes for possible data. The proposed model provides the optimal number of replicas to minimize the overall network traffic in both cases.

A node that serves as a rendezvous (replication) point, will experience a higher traffic load associated with supporting consumption and production, and thus its energy reserves will be depleted at a higher rate. This is also the case for nodes that serve to transport information among replication points. Thus, it is desirable to balance such roles among the network's nodes. To this end, application's timeline is subdivided into *epochs*. During each epoch a new set of replication nodes is randomly selected. Moreover, in each epoch one can, not only choose a new set of replication points, but also adapt the number of replicas to match changes in an application's production and consumption traffic. The proposed framework is thus highly flexible, yet also presents challenges towards optimizing its adaptation to application's traffic.

Related work. We have presented in [4] a detailed survey that discusses the main works on Data-Centric Storage (DCS) covering different aspects. Due to the lack of space we only mention in this section those works that are closely related to the contributions of this paper.

The key ideas underlying DCS were first presented in [1] where the authors introduced a Geographic Hash Table (GHT) as the first DCS system. In this proposal a single replication node is used.

Approaches using multiple rendezvous (replication) nodes have been previously proposed [2, 3, 4, 5, 6], yet these studies place replicas in a structured, i.e., on grids or uniform manner, as opposed to our approach based on selecting random locations. For instance, the authors of GHT proposed to create a grid-structured replication mechanism [2, 3] (GHT with multiple replicas), in which the number of cells in the grid follows a geometric formula 4^d, where d is the so-called *network depth*. Thus the number of replicas grows exponentially as 1, 4, 16, 64, etc., which can lead to poor performance due to the coarse granularity of changes in d. Moreover, these works do not discuss anything regarding the appropriate number of replicas to be used.

Tug-of-War(ToW) [5] follows the same grid-structured replication mechanism proposed by GHT with multiple replication nodes. However they provide two main contributions: (i) a mathematical model to calculate the optimal *network depth*, d, based on the application consumption and production traffic; (ii) the so-called, *combing routing*, that takes advantage of the grid replication fashion to provide a more efficient routing to allow replication nodes to communicate to each other.

In [4] we have presented a Quadratic Adaptive Replication (QAR) system that is more adaptive than ToW and GHT with multiple replication nodes. It is also a grid-based replication scheme, but it calculates the number of replicas as, $N_r = d^2$, which allows the number of replicas to grow in a quadratic fashion as 1, 4 , 9, 16, 25, 36, etc. We also provide a mathematical model that leads to the optimal number of rendezvous nodes to be used based on the consumption and production traffic. We demonstrate that QAR outperforms ToW and by extension GHT with multiple replicas due to its greater adaptivity,.

In [6] the authors present a theoretical framework that defines the Scaling Laws for DCS in terms of energy burdens and storage. They also provide a mathematical model that calculates the optimal number of uniformly deployed replication nodes to be used in the sensornet. However, they do not validate that theoretical model and as we will demonstrate in section 4, using the number of replicas suggested by this paper leads to a much worse performance than ToW, QAR and random replication.

Most of the abovementioned works assume a square sensor field. If the sensor field is not square, e.g. rectangular or some other irregular shape, the approaches in [2, 3, 4, 5], could become much less efficient. By contrast, our approach using random replication is easily adapted to any sensor field shape, as long as the shape is known a priori by network's nodes. Specifically random locations can be generated until the right number lie inside the region of interest

Therefore, random replication is not only simpler and more flexible than previously proposed approaches, but, also, as will be demonstrated in the sequel, enables an effective reduction of network traffic relative to previous work.

The idea of changing the DCS rendezvous point over the time has been mentioned in [7, 8, 9]. However, it is just mentioned, these works do not analyze which are the cost and implications of such a change and how it affects the network performance.

Contributions. To the best of our knowledge, this paper makes several novel contributions to the study of Data-Centric Storage for Wireless Sensor and Actuators Networks (WSAN). First, we propose a generic replication framework using sets of randomly located replicas that can change over the time. Second, we propose and validate a simple model to determine the optimum number of randomly-placed replicas, in terms of minimizing the overall traffic and associated transmission energy, given the measured intensities for production and consumption of an application. Third, we propose a simple mechanism to equalize the energy burdens across the network and to adapt the degree of replication to an application's (possibly changing) traffic and the energy burdens on the network. We achieve this by changing replicas over the time. An analysis of the implications of changing replication nodes is also presented in this paper. Moreover, we demonstrate that changing the set of randomly located rendezvous nodes extends the WSAN lifetime at least by 60% when compared to previous proposals in the literature. This enhancement is shown to reach factors of 10x under some conditions. Finally we propose various mechanisms to implement the above information delivery framework. In particular we propose the use of a *Meta-Information Service* in a WSAN supporting multiple applications. This service enables efficient bootstrapping of new sensor nodes and new applications, while addressing key fault-tolerance requisites for such networks.

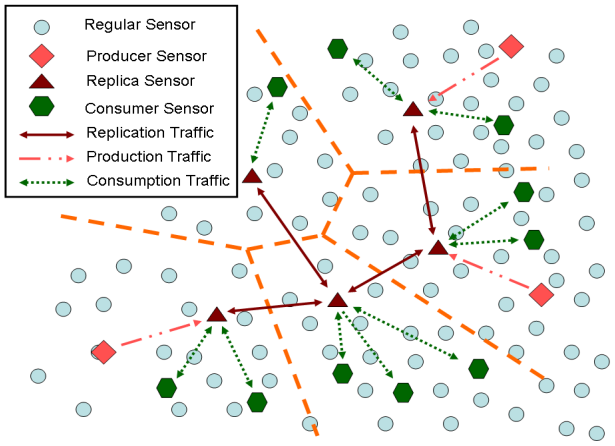


Figure 1: Example of Data-Centric Storage with 5 randomly-placed replicas.

Paper Organization. The remainder of this paper is structured as follows: Section 2 describes our assumptions and explains in more detail the basic framework operation. The analytical model employed to analyze and optimize resource utilization is presented in Section 3. In Section 4 we compare the performance of random replication versus previous proposals in the literature and analyze the benefits and performance of changing replicas over the time. Section 5 focuses on advanced protocol design considerations to efficiently realize the proposed replication framework. Finally, Section 6 offers concluding remarks and discusses the promise of the proposed approach.

2. BASIC SYSTEM OPERATIONS

We begin by summarizing the main assumptions made in this paper. The focus is on distributed applications operating autonomously over a sensor and actor network without external intervention or communication. The name of an application is known by all the consumer/producer nodes that are part of the application. The production events and consumption interest associated with a given application will be assumed to be roughly spatially homogeneous. We consider a static wireless sensor and actor network that involves a large number of homogeneously distributed nodes, which transport information by relaying packets across neighboring nodes. Nodes are assumed to know their spatial location within the operational area as well as the network dimensions and realize a geographic routing service (e.g. [10]) that is able to unambiguously route packets to the closest node to a given spatial location.

Below we shall introduce basic functionalities required in our proposed framework for the case where consumption dominates production traffic, see e.g., Fig. 1. Let's consider that the application's name is APP, the current epoch is e and, based on the current ratio of consumption to production demand (λ_c/λ_p) and the network dimensions, the optimal number of replication nodes is N_r (this will be discussed in Section 3). To simplify the description, we start assuming that this info is known by every application's node.

Producers and consumers functionality. Suppose a producer or consumer node generates an event or queries some data (respectively) related to APP. It then determines the closest replication point by computing the Euclidean distance between its spatial location and that of all replication points obtained from the hash operation: $hash(APP \oplus e \oplus i)$, i=$\{0, 1, ..., N_r - 1\}$. Once the producer/consumer node determines the closest rendezvous point, it forwards a message/query to that location, i.e., to the node n_0 closest to the location. In case of consumption, the rendezvous node just responds with suitable data to the query. This replication location will be used for some time, so the producers and consumers may cache the replication point coordinates avoiding recomputation for every single message[1].

Creating a tree to replicate data over rendezvous nodes. The next step is creating a *minimum spanning tree* rooted at n_0 over which data replication takes place. Each replication node needs to determine the set of nodes (if any) to which it should forward new data. Since all rendezvous nodes know the hashed locations, we will consider without loss of generality, the construction of the replication tree from the point of view of any given rendezvous node, e.g., the root node. The root node, n_0, manages three sets of replication nodes:

- $\mathcal{C}$: the set of rendezvous nodes already covered by the replication tree, where initially $\mathcal{C} = \{n_0\}$.
- $\mathcal{R}$: the set of rendezvous nodes to be reached, which initially contains all the rendezvous nodes except the root node: $\mathcal{R} = \{n_1, n_2, ..., N_{r-1}\}$.
- $\mathcal{F}$: the set of rendezvous nodes to which the current rendezvous node should forward the event, which is initially empty: $\mathcal{F} = \emptyset$

The algorithm proceeds as follows. The root node computes which rendezvous node in $\mathcal{R}$ is closest to n_0. Suppose it is n_1, then n_1 is removed from $\mathcal{R}$ and included in both $\mathcal{C}$ and $\mathcal{F}$., i.e., $\mathcal{C} = \{n_0, n_1\}$, $\mathcal{R} = \{n_2, ..., n_{r-1}\}$, and $\mathcal{F} = \{n_1\}$). Next, it computes the rendezvous node in $\mathcal{R}$ which is closest to anyone in $\mathcal{C}$. If the closest distance is between the root n_0 and n_2, then n_2 is removed from $\mathcal{R}$ and included in $\mathcal{C}$ and $\mathcal{F}$. However, if the closest distance is the one between n_1 and n_2, n_2 is also removed from $\mathcal{R}$, but only included in $\mathcal{C}$. The process is repeated until $\mathcal{R}$ is empty, at which point $\mathcal{F}$ contains all the forwarding rendezvous nodes of n_0. Assuming each node knows who the root is, each one can similarly compute the associated forwarding sets $\mathcal{F}$. Note that if the above distributed mechanism is used, there will be one replication tree per rendezvous node, which serves as its root. The routing table of a replication node associated with a given application would have one entry per replication node acting as root node for production events, with the associated forwarding nodes $\mathcal{F}$ obtained after running the algorithm. Alternatively only one of these trees could be shared among all replicas.

[1]In this section we will equivocate the rendezvous nodes with the associated hashed locations. There are several ways of finding the closest node to a given location, but they are energy consuming. Thus, we consider that the first time a rendezvous node is contacted by other node, it notifies to that node its actual location, so from that moment the contacting node can directly communicate with the rendezvous node avoiding the energy expenditures of finding the closest node to a given location for each message.

Changing the set of rendezvous nodes. We define an *epoch* as the time between two consecutive changes in the set of replication nodes. In addition, we consider two events that could trigger epoch changes: (i) when a node serving as a replication node exceeds a threshold in the number of messages sent and received since the epoch started; and, (ii) just before one such node runs out of battery. Whichever happens first triggers a change of epoch.

At the beginning of each epoch, a rendezvous node gathers local traffic statistics (number of messages sent and received, traffic intensity in bits/sec, etc) during a predefined time interval Δt. After that time, each rendezvous node broadcasts over its replication tree (using piggybacking in data packets or dedicated control messages) its local production/consumption traffic measurements and its estimate for the residual time for the epoch, to the remaining replicas. In turn, based on the exchanged estimates, each replication node computes the minimum estimate for the epoch's residual time, along with the number of rendezvous nodes that should be used in the next epoch, based on the overall measured traffic.

When the estimated epoch deadline arrives, current rendezvous nodes know the locations of the current set, and can compute the locations of the (possibly different) number of nodes in the set for the next epoch using the epoch-dependent hash function. Now each current rendezvous node, need only to determine if it is the closest node to one of the nodes in the subsequent set. If so, such nodes can directly transfer in parallel their stored data to the new locations.

Consistent notification of epoch changes to producers and consumers. Once the current set of rendezvous nodes decides on the next epoch change, consumers and producers need to be notified when it will be initiated and the number of replicas to be used. This can be achieved as follows. At the beginning of an epoch, active consumers and producers set a flag in their messages. This flag indicates to the replication node that this particular consumer or producer does not yet know the current epoch duration nor the number of replicas for the next epoch. After Δt when the current replication nodes have estimated both values, they send a message or piggyback this information back to producers and consumers, respectively. Consumers and producers receiving the information can then cancel the flag until the start of the next epoch. This simple and robust mechanism does not require rendezvous nodes to know who the producers and consumers are, thus saving memory and enabling scalability. By proactively predicting and sharing information about epoch changes it enables consumers and producers to experience a smooth epoch transition.

3. SYSTEM MODEL

In this section we propose a simple stochastic geometric model for the network that permits optimization of the large scale system's parameters, i.e., intensity of replication nodes. The approach follows the seminal work of [11, 12] and our own work in applying this methodology to ad hoc wireless networks, e.g., [13, 14].

The locations of nodes in the Wireless Sensor and Actor Network are assumed to be fixed, and modeled by a homogeneous spatial Poisson Point Process Π_n, i.e., a 'random' set of points on the plane, with intensity λ_n locations per unit area [15]. A fraction of those nodes are randomly, independently sampled to serve as replication nodes. Under these conditions the replication nodes also follow a homogeneous spatial Poisson Point Process Π_r, with intensity $\lambda_r < \lambda_n$. Production and consumption events, generated by some networks nodes, are in turn modeled by independent homogeneous spatio-temporal Poisson Point Processes Π_p and Π_c each with intensities λ_p and λ_c events per unit time and unit area respectively. To avoid unnecessary complications, we shall assume that spatial process Π_r and spatial temporal point processes Π_p and Π_c are mutually independent. Note this is not the case in reality, since they are connected through the locations of the nodes Π_n in the network. However if λ_n is high, the impact on our model is minimal– we shall verify this via simulation in the sequel. Although the model corresponds to one on an infinite plane we shall restrict attention to a fixed region $\mathcal{A} \subset \mathbb{R}^2$ modeled as a convex set with area $A = |\mathcal{A}|$, and optimize operation on $\mathcal{A}$ roughly ignoring edge effects. On average there are $N_r = \lambda_r A$ replication nodes in $\mathcal{A}$.

3.1 Evaluating overall network traffic and energy costs.

Let us first consider the overall network traffic generated by consumption and production events inside the network. The overall metric here is the total traffic load, measured in bits-m/sec that need to be supported by the network, i.e. in region $\mathcal{A}$. Recall that in an ad hoc wireless network traffic load can not simply be measured in terms of bits/sec, but must also account for the distance packets must travel, since this involves relaying, and thus resources along the path. Measuring network load in terms of bits-m/sec captures the amount of traffic and the distance that must be traveled. In turn, we assume the power expenditures for transporting traffic to be roughly proportional to the overall network traffic measured in bits-m/sec.

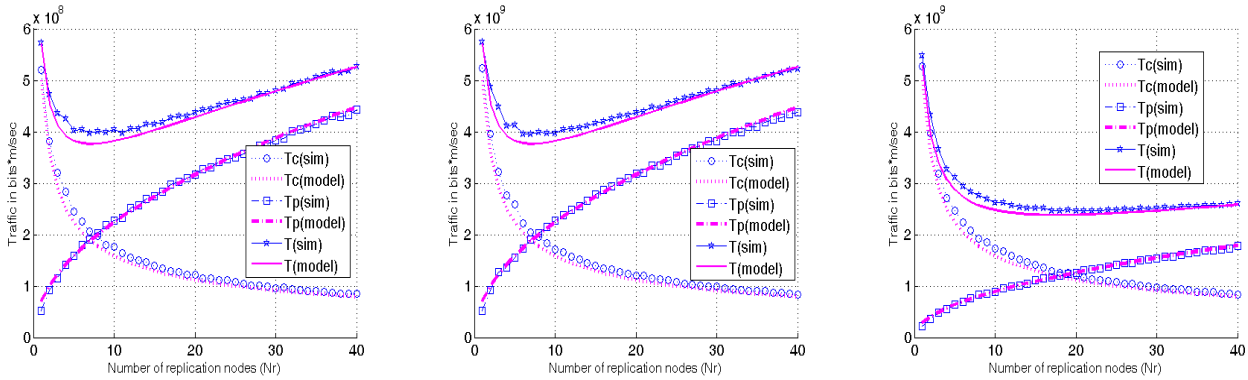
Case 1: Consumption dominates production ($\lambda_c > \lambda_p$)

We assume consumers retrieve data from the closest replication node. Thus consumption events can be partitioned based on the Voronoi tessellation [12] induced by the replication nodes. The average size of such cells is $1/\lambda_r$, the mean number of consumption events in such a region per unit time is λ_c/λ_r. Meanwhile the typical distance from a consumer to its nearest replication node can be shown to be $\frac{1}{2\sqrt{\lambda_r}}$ [11]. Thus the total consumption traffic, $T_c(\lambda_r)$, for the region $\mathcal{A}$ is proportional to the number of replication nodes $\lambda_r A$, times the number of consumers per replication node cell λ_c/λ_r, further multiplied by the mean distance between consumers and replication nodes $\frac{1}{2\sqrt{\lambda_r}}$, i.e.,

$$T_c(\lambda_r) = \alpha \lambda_r A \frac{\lambda_c}{\lambda_r} \frac{1}{2\sqrt{\lambda_r}} = \alpha A \frac{\lambda_c}{2\sqrt{\lambda_r}} \quad \text{bits-m/sec},$$

where α is a proportionality constant corresponding to the average number of bits per consumption event that are exchanged between the consumer and its nearest replication node.

Next, we consider the replication cost when new data is produced. Again new data is produced on our network at a rate $\lambda_p A$ events per unit time. We shall assume that data associated with each new event is distributed to the replication points in the network along a *radial spanning tree* [16] which includes all the replication nodes. The total length per unit area for radial spanning trees over a homogeneous



(a) λ_c=$50*10^{-6}$;λ_p=$10*10^{-6}$;N_r^*=7.07 (b) λ_c=$500*10^{-6}$;λ_p=$100*10^{-6}$;N_r^*=7.07 (c) λ_c=$500*10^{-6}$;λ_p=$40*10^{-6}$;N_r^*=17.67

Figure 2: Consumption, production and overall traffic generated by using different number of replication nodes (A=1000x1000, N=5000, α=200 bits, β=100 bits).

Poisson Point Process can be computed and is indeed very close to that of a minimum cost spanning tree. In particular for a large disc of radius x, the total length for a radial spanning tree centered at the origin grows as $\frac{\pi x^2\sqrt{\lambda_r}}{\sqrt{2}}$, so the average length of the tree per unit area is given by $\sqrt{\lambda_r/2}$ [16]. The total production traffic generated, $T_p(\lambda_r)$, is thus given by β bits per event, times the rate of production events $\lambda_p A$ in the network, times the length of the associated radial spanning tree:

$$T_p(\lambda_r) = \beta\lambda_p A\sqrt{\frac{\lambda_r}{2}}A = \beta A^2\lambda_p\sqrt{\frac{\lambda_r}{2}} \text{ bits-m/sec.}$$

Note that we have assumed for simplicity that the radial spanning tree is rooted at the location where the event is produced. Alternatively one could assume that the new event is first transported to the nearest replication node that then employs a radial spanning tree to reach the remaining replicas. The replication cost in this second case has a similar scaling.

The total network traffic, $T(\lambda_r)$, is thus given by:

$$T(\lambda_r) = \alpha A\lambda_c\frac{1}{2\sqrt{\lambda_r}} + \beta A^2\lambda_p\sqrt{\frac{\lambda_r}{2}} \text{ bits-m/sec.}$$

We can optimize this to obtain an optimal spatial intensity for replicas λ_r^* given by:

$$\lambda_r^* = \frac{\alpha\lambda_c}{\sqrt{2}\beta A\lambda_p} \text{ replicas/m}^2,$$

and the associated minimum overall network traffic is given by:

$$T(\lambda_r^*) = 2^{1/4}\sqrt{A}\sqrt{(\alpha\lambda_c A)(\beta\lambda_p A)} \text{ bits-m/sec.}$$

REMARK 1. ***Scaling characteristics.*** *Roughly speaking the optimal average number of replicas for the network covering an area A is given by:*

$$N_r^* = \lambda_r^* A = \frac{\alpha\lambda_c}{\sqrt{2}\beta\lambda_p}. \quad (1)$$

Surprisingly, this only depends on the ratio of the intensity of consumption to production. Thus if one were to double the intensity of consumption and production for a fixed area, the same number of replicas would be optimal. If however one stretches the area by a factor of two, this would decrease the intensity of production and consumption by 2, maintaining the same ratio, yet the optimal intensity λ_r^* per unit area would also have to decrease by a factor of 2. Furthermore we note that the overall network load, in bits-m/sec scales as $\sqrt{A}$ times the geometric mean *of the total rate of consumption, $\alpha\lambda_c A$ in bits/sec and the rate of production $\beta\lambda_p A$ in bits/sec. This gives a sense of the growth of overall traffic with network size.*

In order to validate this model we have simulated random realizations of the network and obtained the consumption (T_c), production (T_p) and total network cost (T) for different numbers of replicas. Unless otherwise stated, all results correspond to at least 50 simulations of different network realizations where $N = 5000$ nodes are randomly placed in a 1000×1000 region. We set β=100 bits, assuming that producers periodically send the information to the closest replica without any acknowledgment. We set α=200 bits since we assume that a consumer first sends a query message to its closest replica and then receives a reply from it. We show 90% confidence intervals on all graphs unless they are so small that they cannot be distinguished.

Figure 2 exhibits the overall consumption, production and total traffic measured in bits-m/sec obtained by the model and by simulation for three different (λ_c,λ_p) pairs: ($50*10^{-6}$,$10*10^{-6}$), ($500*10^{-6}$,$100*10^{-6}$) and ($500*10^{-6}$,$40*10^{-6}$) $\frac{bits}{sec*m^2}$. The number of replicas employed varies from 1 to 40. Thus, the optimal average number of replicas for these cases is 7.07, 7.07 and 17.67 respectively. Figures 2(a) and 2(b) illustrate the scaling properties of the framework versus the ratio of consumer to producer intensities. Note that both scenarios have exactly the same optimal number of replicas, even though the latter's application generates ten times more production and consumption events than the former. It is worth noting that for applications with a high λ_c/λ_p ratio (see Figure 2(c)), there are several values around the optimal number of replicas that could be employed instead, because they generate a similar overall traffic.

It must be highlighted that this simple model establishes traffic metrics assuming routes follow straight lines. However, WSANs which are the focus of this paper are multihop networks where routes unlikely follow straight paths.

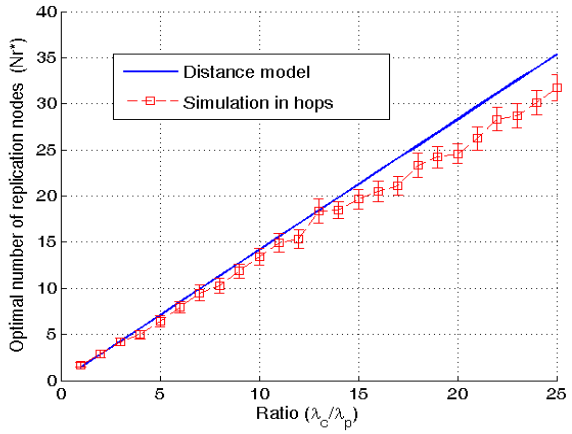


Figure 3: Optimal number of replicas that minimizes the overall number of messages (A=1000x1000, N=5000, Tx=50 m)

To that end we have verified that for networks that have a sufficiently high density of nodes, the optimal number of replicas obtained by our idealized model reflects the actual optimal number of replicas on a given network. For this purpose we have simulated a sensor network employing greedy forwarding [10] and a transmission range $Tx = 50$ m. We considered a setup where the ratio λ_c/λ_p varied from 1 to 25. Figure 3 shows the number of replicas that minimizes the overall simulated traffic based on the actual number of hops of all messages versus the optimal number of replicas suggested by our model. As it can be seen, when there is a low number of replicas, the model and the simulations are a good match. A small discrepancy occurs for high λ_c/λ_p ratios. However, as mentioned earlier, in the case this ratio is high, the overall cost is not very sensitive to the precise optimal value for the number of replicas.

Case 2: Production dominates consumption ($\lambda_c < \lambda_p$)

If the intensity of consumption is low relative to that of production it may be preferable not to copy data across all replication points. Instead producers can store data solely at the closest replication node. Subsequently consumers should contact *all* replication points to gather the full information. This could be done in several ways[2] although we will consider the simplest one where consumers contact all the replication points directly.

In this case the overall production traffic is:

$$T_p(\lambda_r) = \beta A \frac{\lambda_p}{2\sqrt{\lambda_r}} \quad \text{bits-m/sec.}$$

The consumption cost can be modeled using the average distance between any two nodes of the network $\sqrt{A}/2$, as the distance from a consumer to each replica, times the number of consumers and replicas. Thus the overall consumption traffic is given by:

$$T_c(\lambda_r) = \alpha(\lambda_c A)(\lambda_r A)\frac{\sqrt{A}}{2} \quad \text{bits-m/sec.}$$

The total network traffic is then given by:

$$T(\lambda_r) = \alpha\lambda_c\lambda_r A^2 \frac{\sqrt{A}}{2} + \beta A \frac{\lambda_p}{2\sqrt{\lambda_r}} \quad \text{bits-m/sec.}$$

One can again find the optimal replication λ_r^* for this case, which is given by:

$$\lambda_r^* = \frac{1}{A}\left(\frac{\beta\lambda_p}{2\alpha\lambda_c}\right)^{2/3} \quad \text{replicas/m}^2.$$

The associated minimum overall network cost is:

$$T(\lambda_r^*) = (\beta\lambda_p)^{2/3}(2\alpha\lambda_c)^{1/3}\frac{3A\sqrt{A}}{4} \quad \text{bits-m/sec.}$$

Note that in this regime the optimal intensity for replicas is a more 'complex' function, i.e., cubic of the ratio of production to consumption intensities, yet, in principle, still easily computable by sensors in the field. This model has been also validated via simulation. However due to space limitations we can not include the validation graphs, which are similar to the ones of the previous section.

3.2 Cost of changing the set of replication nodes to balance network loads.

We have argued that it would be worthwhile to periodically change the set of nodes where data is replicated. The cost of moving from one set of replica nodes to another should be relatively low since this is a highly-parallel distributed process. In particular, suppose the current intensity of replicas is λ_r^c and we wish to move to a new set of randomly-located replicas with intensity λ_r^n. Note that the new set of replicas need not to have the same intensity as the current one. Also suppose each replica node currently holds an average amount of data d.

A rough estimate of the energy cost associated with moving data from the current set of replication nodes to the new one $T_r(\lambda_r^c, \lambda_r^n)$, can be evaluated as follows. Each old replica would contact one of the new nodes. Given that the distance to a new randomly located replica from one of the current nodes is $\frac{1}{2\sqrt{\lambda_r^n}}$ the total cost in a network of area A would be roughly:

$$T_r(\lambda_r^c, \lambda_r^n) = \frac{d}{2}\frac{\lambda_r^c A}{\sqrt{\lambda_r^n}}. \text{ bits-m}$$

So if $\lambda_r^n = \lambda_r^c$ the cost is $T_r = \frac{d}{2}\sqrt{\lambda_r}$ bits-m. If the set of replication nodes changes infrequently, then the contribution to the overall network traffic and energy consumption of changing the set of replicas would be fairly small. However this does depend on λ_r and the frequency of such updates. We shall consider this in more detail in the next section.

4. PERFORMANCE EVALUATION

In this section we consider two questions: (1) how selecting rendezvous nodes' locations at random compares to previous grid-based and uniform-based proposals; and (2), whether it is worthwhile to change the set of rendezvous nodes over time considering the associated overheads. We have developed a custom simulator that provides more scalability than standard ones, since it does not simulate wireless

[2]A symmetric model to that presented in the case of consumption dominating production could be also proposed. However, that model would assume that both, queries and replies, are sent through the replication tree once per branch. This can only be achieved if replies are aggregated and such aggregation has implications that are out of the scope of this paper.

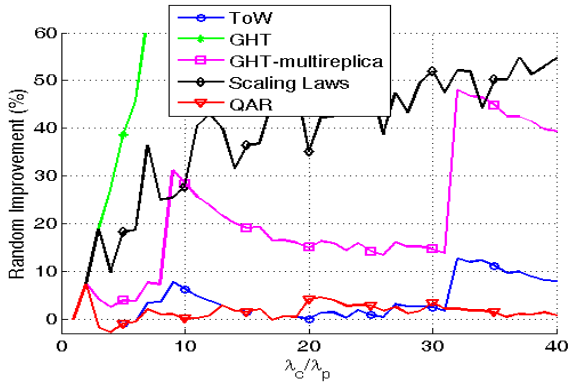


Figure 4: Improvement of overall traffic when using Random vs. ToW, QAR, Scaling-Laws, GHT and GHT with multiple replicas (A=1000x1000, N=5000, Tx=50 m, α=200 bits, β=100 bits)

.

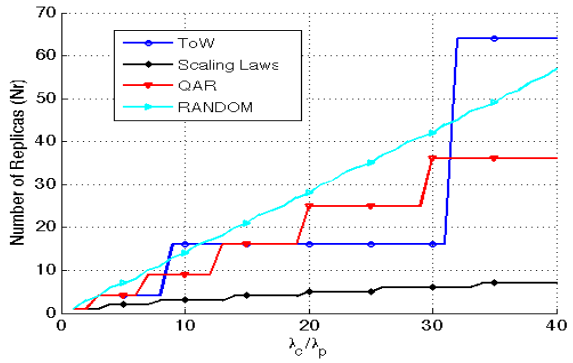


Figure 5: Number of replicas used for the different proposals: ToW, QAR, Scaling-Laws and Random (A=1000x1000, N=5000, Tx=50 m, α=200 bits, β=100 bits)

.

communications (i.e. PHY and MAC) other than transmission range.

4.1 Random vs. grid-based and uniform replica allocation

We have compared our work with those that are similar in spirit: ToW [5], QAR [4], Scaling Laws [6], the original GHT proposal [1] that uses a single replication node and GHT with multiple replication nodes [2, 3]. For the last case, since the authors do not propose any way to obtain the number of replicas to be used, we select the same number used in ToW since both works are grid-based and uses the same 4^d geometric formula for the number of rendezvous nodes.

In order to compare these approaches, we have run simulations for a large WSAN with the following characteristics: an area $A = 1000x1000\ m^2$, $N = 5000$ sensors, transmission range $Tx = 50$ m and λ_c/λ_p ranging from 1 to 40. For each λ_c/λ_p ratio we have simulated 50 scenarios to estimate the mean network cost realized by the different replication approaches. In order to get meaningful results, we use the number of hops traversed as the measure of the overall traffic cost.

Figure 4 shows the network traffic improvement achieved using random replication compared to all the other approaches and Figure 5 depicts the number of replicas used by each approach for each particular λ_c/λ_p ratio.

Random replication reduced the overall traffic by an average of 137% compared to GHT, 39% compared to Scaling Laws, 21% compared to GHT with multiple replication nodes, 4% compared to ToW and 1.5% compared to our previous QAR proposal. Moreover, this improvement reaches peaks around a 50% when compared to Scaling Laws and GHT with multiple replicas, 15% to ToW and 7% to QAR.

The main reason our solution achieves a better performance, is that random replication allows a much smoother range over which to adapt its evolution. That is, the optimal number of replicas grows linearly, whereas ToW and GHT with multiple replicas employ a 4^d geometric growth and QAR a quadratic one (see Figure 5). For instance, in some cases ToW must choose between 16 or 64 replicas, where none of them is a good fit for the scenario of interest.

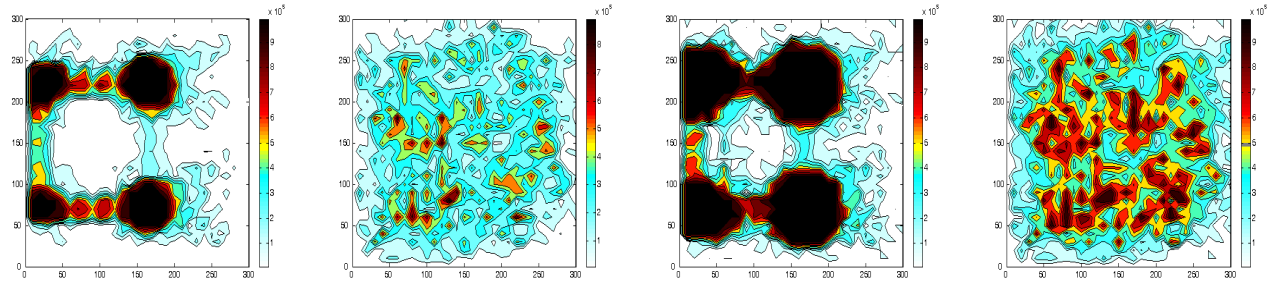
Therefore, random replication is demonstrated to be the one minimizing the overall network traffic improving all previous approaches in the literature that use grid-structured or uniform replication. Moreover, processing the random locations for the rendezvous nodes is easier to be adapted for different sensornet shapes than those using structured replication. Hence, random replication is simpler and more cost-effective.

4.2 Changing Replicas over the time

Sensors selected as rendezvous nodes (and those close to them) will naturally expend more energy than other nodes. Thus, if the responsibilities of nodes do not change, those nodes are most likely to run out of energy reserves. In [1, 2, 3, 4, 5], when this happens, an alternate sensor close to the previous replication point is selected as the new rendezvous node, until its battery expires, and so on. After some time, routing (and sensing) holes will be created around the original replication coordinates, affecting the routing of the network.

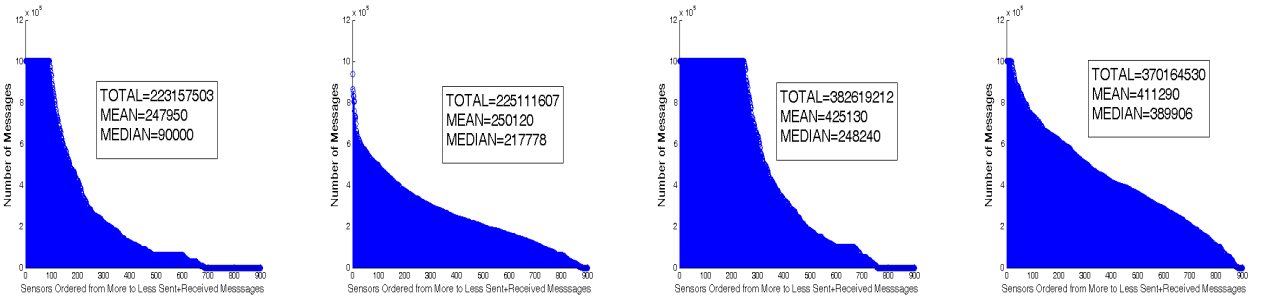
On the other hand, if replication points change over time, the extra energy expenditures associated with rendezvous nodes can be balanced across all the nodes in the network, thus extending the network's lifetime, and avoiding the creation of routing holes. In addition, although moving replication points has an associated overhead, this does not mean that network energy expenditures become higher than keeping rendezvous nodes static. Indeed, when replication nodes are kept static, longer paths will be required for communication, which in turn will consume more energy. In this section we demonstrate that routing holes can have more impact on the overall network energy expenditures than the cost of changing the set rendezvous nodes.

In order to verify the abovementioned points, we ran simulations comparing ToW [5] using static replicas with random replication where the set of replication nodes change over the time. We use a grid-based node deployment (which makes energy maps generation easier) with $N = 900$ sensors, over a square of area $A = 300x300$ m^2. Each sensor has a transmission range $Tx = 30$ m. In addition, due to space constrains, we only show the case where consumption dominates production ($\lambda_c > \lambda_p$). We use the number of



(a) ToW-Static after 30K cycles (b) Random-Epochs after 30K cycles (c) ToW-Static after 50K cycles (d) Random-Epochs after 50K cycles

Figure 6: Energy map of the number of messages sent and received by all sensors of the network (A=300x300, N=900, Tx=30, N_p=100, N_c=300, L=10).



(a) ToW-Static after 30K cycles (b) Random-Epochs after 30K cycles (c) ToW-Static after 50K cycles (d) Random-Epochs after 50K cycles

Figure 7: Distribution of the number of messages per sensor (A=300x300, N=900, Tx=30, N_p=100, N_c=300, L=10).

messages in the network as a first order proxy for consumed energy. A sensor node's energy is depleted once it sends and/or receives one million messages. Finally, the threshold that determines the end of an epoch, E_{th}, is set to 300000 messages (30% of battery) [3].

For these simulations we have used geographical routing based on greedy forwarding. When greedy forwarding fails, i.e. due to routing holes, we use the optimal path from the node where the greedy forwarding stopped to the destination node.

Time is measured in cycles in order to scale the simulations and be able to deploy a large number of sensors. A cycle is a time period in which every consumer node performs one consumption event and every producer node generates a production event. Since energy is measured in terms of messages, the traffic is measured in $messages/cycle$[4]. We deploy 300 consumers (N_c), which means 300 queries and 300 replies per cycle, and 100 producers (N_p) that generate 100 production events per cycle. The consumption to production traffic ratio results in an optimal number of replicas equal to 4 for both ToW and random replication

In order to measure the cost of an epoch change, we assume that the produced data has a mean lifetime of L cycles. Then, since the production intensity is λ_p and the network area is A, the average data at each replication node is $d = \lambda_p AL = N_p L$ messages. L is set to 10 cycles for these simulations, thus the replication change is costly, because it means that 10 messages per producer are moved from the old replicas to the new ones. Having 100 producers, this means a total cost of 1000 messages. .

Figure 6 shows the energy distribution map after a simulation time of 30000 and 50000 cycles. Figure 7 shows the number of messages sent+received by each sensor at the same cycles, as well as the mean and median values per sensor and the total messages sent and received in the whole network, that roughly captures the total energy consumed by the network.

As seen in Figures 6(a) and 6(c), keeping static replication points creates routing holes in the network, with 93 and 247 expired sensors after 30000 and 50000 cycles respectively. The number of depleted sensors are only 0 and 17 respectively, when replication nodes are changed over the time. Furthermore, simulation results obtained later in time (70959 cycles) show that the static network is eventually disconnected, because holes become very large and coalesce. In addition, more sensors participate of the network operation when epochs are used. As shown in Figure 7, all sensors except 18 (2%) after 30000 and 15 (1.7%) after 50000 cycles, have sent and/or received at least one message, whereas in the case of static replication nodes more than 200 (22%)

[3]We also ran experiments for $E_{th} = 100000$, $E_{th} = 500000$ and $E_{th} = 700000$ and all of them clearly outperformed the static solution.

[4]The production (λ_p) and consumption (λ_c) intensities are then measured in $\frac{messages}{cycle*m^2}$

Lifetime Criteria	1^{st} *dead*	*1% dead*	*10% dead*	*25% dead*	*40% dead*	*10% cons+prod*	*25% cons+prod*	*Network disconnection*
ToW-Static (cycles)	2619	7328	29086	47830	63230	31668	47984	70952
Random-Epochs (cycles)	31199	41124	66750	87968	101523	65171	79717	170950
Improvement(%)	1091%	461.2%	129.5%	83.9%	60.6%	105.8%	66.13%	140.9%

Table 1: Sensornet Lifetime ToW-Static vs Random-Epochs

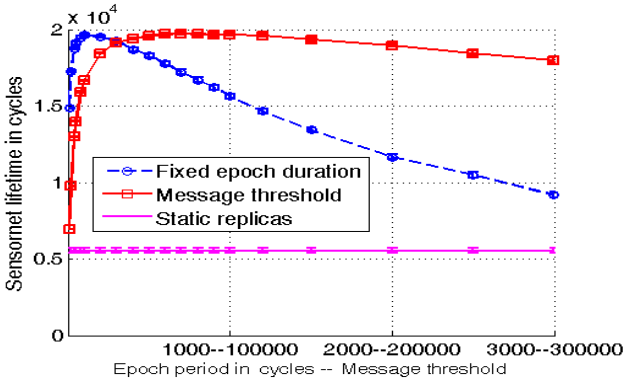


Figure 8: Sensornet lifetime comparison (A=1000x1000, N=5000, Tx=50, L=10, m=5). X axis refers to the cycles for changing the epoch in the fixed duration approach, or the message threshold.

nodes have not sent or received any message after 30000 cycles, decreasing to 160 (17.8%) nodes after 50000 cycles. When considering the total energy consumed by the network, the dynamic approach uses just 0.8% more energy than the static one after 30000 cycles. However, a 3.3% extra energy is required by the static approach after 50000 cycles. This shows that the cost of using longer routing paths eventually exceeds that of changing the rendezvous nodes over time.

In Table 1, we compare the sensor network lifetime using both approaches: static ToW and random replication with epochs. Since lifetime can be defined by different metrics [17] (first sensor running out of battery, some percentage of sensors running out of battery, important nodes like consumers and/or producers running out of battery, some part of the network disconnects and many messages are lost, etc), we provide a broad overview of metrics to let the reader establish a fair comparison depending on the criterion used to define the network's lifetime. The table shows the number of cycles spent until each lifetime criterion is reached. For all the criteria our solution extends the network's lifetime by at least 60%. We note that in many cases changing replicas over the time and using random replication extends the network's lifetime by a factor of 2x.

Finally, Figure 8 shows how using a message threshold to trigger epoch changes compares to employing a fixed epoch duration as proposed in [7]. We simulate a larger (N=5000, A=1000x1000, Tx=50) multi-application sensor network with random replication. We set up m=5 heterogeneous applications with (20, 40), (60, 120), (100, 200), (140, 280) and (180, 360), ($\frac{production-events}{cycle}$, $\frac{consumption-queries}{cycle}$) pairs, calculating the network's lifetime (1% of sensors expire) for both the two dynamic approaches versus using a fixed static set of random located replicas. Again the need of changing replicas over the time versus using static ones is clear. In addition, as seen in the figure, our proposal to trigger epoch changes based on message counts is more robust to the precise setting of the message threshold than using a fixed epoch duration.

5. ADDITIONAL DESIGN CONSIDERATIONS

In order to become a viable solution, our replication framework has to be further developed to address several practical issues. This section provides additional insight on developing a real-world Data Centric Storage protocol with multiple randomly-placed replicas that change over the time. Such a protocol should implement additional mechanisms in order to:

- Provide a bootstrapping mechanism for finding the current set of replicas if the epoch value is unknown.
- Provide fault tolerance in the case replication nodes fail.
- Provide a mechanism to bring new applications online.

In order to solve the bootstrapping problem when a new application node wants to participate as a producer or consumer, we propose employing a Meta-Information service where every application in the network stores its current epoch value and the number of replication nodes currently in use. Once a new sensor acquires this information, it can then ask detailed information to the replicas concerning the time at which the epoch will expire and the number of replication nodes to be used in the next epoch, by using the flag mechanism introduced in section 2. This Meta-Information service is just another application that itself may use the proposed replication framework.

The question now is how a new sensor is able to know the current epoch of the Meta-Information service. A straightforward solution is broadcasting to the network when a Meta-Information epoch change happens (e.g. once per hour/day). Since the number of changes could be arbitrarily low, the energy consumption would be negligible. Then, when a sensor bootstraps it can simply ask any of its neighbors what is the current Meta-Information epoch.

Another aspect that should be taken into account is determining how the Meta-Information service knows that a given application is changing its epoch. The first replication node (i=0) could be the one notifying each epoch change to its closest Meta-Information service replication node, which in turn replicates the new epoch to the remaining Meta-Information replication nodes.

The Meta-Information service can be also employed as a fallback mechanism in case of replication node failure or epoch de-synchronization. If a node fails in accessing its closest replication node for a pre-defined number of times, it tries to contact the remaining replication nodes (sorted by distance) from the current epoch, since these locations can be computed locally by the sensor. In the case the sensor has suffered an epoch de-synchronization, it can contact

the Meta-Information service that replies with the current epoch and number of replication nodes being used for that application.

Finally, we shall define how a new application can be brought online on a sensor network. When one of the replication nodes of the Meta-Information Service receives a query from a bootstrapping producer requesting the epoch and number of replicas of an unknown service, it understands that this application does not yet exist. Therefore it registers the application and assigns to that service a random epoch number and a single replication node. After that, the meta-information node notifies the first application's replication node that the service needs to be started, sharing the initial epoch number with both the replication node and the first producer. From that moment any sensor can start using the new application.

6. CONCLUSIONS AND FUTURE WORK

This paper demonstrates that placing multiple replication nodes at random in a DCS system outperforms previous works in the literature proposing a single rendezvous point, grid-structured or uniform replication nodes deployments. Furthermore, it has been shown that equalizing the energy burdens across the network, by mean of changing replication nodes over the time, achieves a huge improvement in the WSAN lifetime, always above a 60%, being under some conditions a tenfold increment. In addition, epoch transitions, far from being more energy demanding, saves energy in front of static approaches that incurs in a higher energy consumption due to the need of using longer communication paths to avoid the routing holes that appear in the network (and grow) along the time.

Our future work will be focused on networks where the event generation is not homogeneous across the network. This is a very interesting topic that has not been previously studied by any DCS work.

7. ACKNOWLEDGMENTS

This work was partially supported by NSF Award CNS-0509355, AFOSR Award FA9550-07-1-0428 as well as IMDEA Networks Madrid and by the Spanish government through the T2C2 Project TIN2008-06739-C04-01.

8. REFERENCES

[1] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Data-centric storage in sensornets," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 137–142, 2003.

[2] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "Ght: a geographic hash table for data-centric storage," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications.* New York, NY, USA: ACM, 2002, pp. 78–87.

[3] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with ght, a geographic hash table," *Mob. Netw. Appl.*, vol. 8, no. 4, pp. 427–442, 2003.

[4] A. Cuevas, M. Urueña, R. Romeral, and D. Larrabeiti, "Data centric storage technologies: Analysis and enhancement," *Sensors*, vol. 10, no. 4, pp. 3023–3056, 2010.

[5] Y.-J. Joung and S.-H. Huang, "Tug-of-war: An adaptive and cost-optimal data storage and query mechanism in wireless sensor networks," in *DCOSS '08: Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems.* Berlin, Heidelberg: Springer-Verlag, 2008, pp. 237–251.

[6] J. Ahn and B. Krishnamachari, "Fundamental scaling laws for energy-efficient storage and querying in wireless sensor networks," in *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing.* New York, NY, USA: ACM, 2006, pp. 334–343.

[7] N. L. Thang, Y. Wei, B. Xiaole, and X. Dong, "A dynamic geographic hash table for data-centric storage in sensor networks," in *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.* New York, NY, USA: IEEE, 2006, pp. 2168 – 2174.

[8] W.-H. Liao, K.-P. Shih, and W.-C. Wu, "A grid-based dynamic load balancing approach for data-centric storage in wireless sensor networks," *Comput. Electr. Eng.*, vol. 36, no. 1, pp. 19–30, 2010.

[9] J. Ahn and B. Krishnamachari, "Scaling laws for data-centric storage and querying in wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1242–1255, 2009.

[10] B. Karp and H. T. Kung, "Gpsr: greedy perimeter stateless routing for wireless networks," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking.* New York, NY, USA: ACM, 2000, pp. 243–254.

[11] F. Baccelli and S. Zuyev, "Poisson-voronoi spanning trees with applications to the optimization of communication networks," *Operations Research*, vol. 47, pp. 619–631, 1996.

[12] F. Baccelli, M. Klein, M. Lebourges, and S. Zuyev, "Stochastic geometry and architecture of communication networks," *J. Telecommunication Systems*, vol. 7, pp. 209–227, 1997.

[13] S. J. Baek, G. de Veciana, and X. Su, "Minimizing energy consumption in large-scale sensor networks through distributed data compression and hierarchical aggregation," *IEEE Journal on Selected Areas of Communications*, vol. 22, no. 6, pp. 1130–1140, August 2004.

[14] S. J. Baek and G. de Veciana, "Spatial model for energy burden balancing and data fusion in sensor networks detecting bursty events," *IEEE Trans. on Information Theory*, vol. 53, no. 10, pp. 3615–29, October 2007.

[15] D. Stoyan, W. S. Kendall, and J. Mecke, *Stochastic Geometry and its Applications.* J. Wiley & Sons, Chichester, 1995.

[16] F. Baccelli and C. Bordenave, "The radial spanning tree of a poisson point process," *Annals of Applied Probability*, vol. 17, p. 305, 2007.

[17] I. Dietrich and F. Dressler, "On the lifetime of wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 5, no. 1, pp. 1–39, 2009.