A

STARR-DCS: Spatio-Temporal Adaptation of Random Replication for Data Centric Storage

ÁNGEL CUEVAS, Institut Telecom, Telecom SudParis
MANUEL URUEÑA, Univesidad Carlos III de Madrid
GUSTAVO DE VECIANA, University of Texas at Austin
ADITYA YADAV, Indian Institute of Technology at Guwahati

This paper presents a novel framework for Data Centric Storage in a Wireless Sensor and Actor Network (WSAN) that enables the use of a *randomly*-selected set of data replication nodes, which also *change over the time*. This enables reductions in the average network traffic and energy consumption by *adapting* the number of replicas to applications' traffic, while balancing energy burdens by varying their locations. 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, based on measurements of applications' production and consumption traffic. Simple mechanisms are proposed to decide when the current set of replication nodes should be changed, to enable new applications and nodes to efficiently bootstrap into a working WSAN, to recover from failing nodes, and to adapt to changing conditions. Extensive simulations demonstrate that our approach can enhance a WSAN's lifetime by at least a 60%, and up to a factor of 10x depending on the lifetime criterion being considered. The proposed framework was implemented in a testbed with 20 motes, and validated in a small-scale scenario the results obtained via simulation for large WSANs. Finally, we present a heuristic that adapts our approach to scenarios with spatially inhomogeneous consumption and/or production traffic distribution providing further benefits in terms of overall traffic and energy consumption reductions.

Categories and Subject Descriptors: C.2.1 [**Network Architecture and Design**]: Wireless communication, Distributed networks

General Terms: Performance

Additional Key Words and Phrases: 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 modelled as a (randomly) distributed set of *producer* and *consumer* nodes, e.g., sensors or actuators that exchange information by relaying packets across neighboring nodes. We assume that producer and consumer nodes do not have explicit

This work is an extension of the paper *Dynamic Random Replication for Data Centric Storage* published in ACM MSWIM'10 and awarded with the Best Paper Award.
Author's address: A. Cuevas, Wireless Networks and Multimedia Services Department, 9, rue Charles Fourier, 91000, Evry, France. M. Urueña, Department of Telematic Engineering, Avenida Universidad, 30, 28911, Leganés, Spain. G. de Veciana, Department of Electrical and Computer Engineering, 1 University Station C0803 Austin, TX 78712-0240, USA. A. Yadav, Indian Institute of Technology Guwahati, Department Computer Science, Assam, India.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

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 and consumers.

Data-Centric Storage (DCS) [Shenker et al. 2003; Ratnasamy et al. 2002; Ratnasamy et al. 2003] is an elegant solution to this problem. The key idea is to identify a node in the network, which will serve as a rendezvous point between producers and consumers associated with the application. This 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 the 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, thus reducing network traffic, energy overheads and response times, 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 $hash(APP \oplus epoch \oplus i) \quad \forall i \in \{1, N_r\}$, where APP is the application's name and *epoch* is a shared time identifier employed to change replicas over the time. The network nodes that are the closest to these hashed spatial locations serve as rendezvous (or replication) nodes for that application. In this setting any producer 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 to the 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 minimize the total network traffic, in bits·meter/second, 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 copied 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 enables the selection of an 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 all of the network's nodes. To this end, the 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 pro-

posed framework is thus highly flexible, yet also presents challenges in terms of optimizing adaptation to application's traffic.

We have implemented the proposed framework on a set of 20 motes for the case when consumption dominates production. Our implementation validates in a real deployment most of the outcomes expected based on our theoretical work.

Finally, to the best of the authors knowledge, all previous work in the literature proposing the use of a replication DCS system assumed that consumption and production intensities are homogeneously distributed across the network. However, this is a very unrealistic assumption for many applications that concentrate major portion of production/consumption traffic in some particular area of the network. Therefore, in this paper we also propose a heuristic that adapts the proposed framework to scenarios in which the production event and consumption query traffics are spatially inhomogeneous.

Related work. In [Cuevas et al. 2010] we presented a detailed survey discussing the main work on Data-Centric Storage. In this section we discuss only related work that is closely related to the contributions of this paper.

The key ideas underlying DCS were first presented in [Shenker et al. 2003] where the authors introduced Geographic Hash Table (GHT) as the first DCS system. This paper considers the use of a single replication node.

Approaches using multiple rendezvous (replication) nodes were subsequently proposed [Ratnasamy et al. 2002][Ratnasamy et al. 2003][Cuevas et al. 2010][Joung and Huang 2008][Ahn and Krishnamachari 2006], yet these studies place replicas in a structured manner, e.g., on a grid, as opposed to our approach based on selecting random locations. For instance, the authors of GHT proposed the creation of a grid-structured replication mechanism (GHT with multiple replicas) [Ratnasamy et al. 2002; Ratnasamy et al. 2003], 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, 256, etc., which can lead to poor performance due to the coarse granularity of changes in d. Moreover, this work does not discuss how to find the appropriate number of replicas to be used.

Tug-of-War(ToW) [Joung and Huang 2008] follows the same grid-structured replication mechanism proposed for 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) and the so-called, *combing routing*, that takes advantage of the grid replication structure to provide a more efficient routing to allow replication nodes to communicate among each other.

In [Cuevas et al. 2010] we proposed 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 defines 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 [Ahn and Krishnamachari 2006] 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 [Ratnasamy et al. 2002][Ratnasamy et al. 2003][Cuevas et al. 2010][Joung and Huang 2008], could become much less efficient. By contrast, the approach proposed in this paper using random replication is easily adaptable to any sensor field area, as long as the shape is known a priori by

the 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 [Thang et al. 2006], [Liao et al. 2010] and [Ahn and Krishnamachari 2009]. However, these works do not analyze what are the cost and implications of such changes and how it affects the network performance, as done in this paper.

Contributions. To the best of our knowledge, this paper makes several novel contributions to the study of Data-Centric Storage for Wireless Sensor and Actuator Networks. First, we propose STARR-DCS, a Spatio-Temporal Adaptation of Random Replication framework for Data-Centric Storage, which uses 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 energy consumption, 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 via simulation that changing the set of randomly located rendezvous nodes in a large WSAN extends the 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. Fourth, we propose various mechanisms to implement STARR-DCS. In particular we propose the use of a *Meta-Information Service* in a WSAN supporting multiple applications. This service enables efficient bootstrapping of new nodes and new applications, while addressing key fault-tolerance requisites for such networks. Fifth, we have implemented STARR-DCS in a 20 mote testbed, obtaining results that validate the main analytical model and simulation results presented in this paper, and show that they are also applicable in a small real network. In addition, the implementation demonstrates that STARR-DCS can easily be implemented on commercial motes. Finally, we extend our baseline work (and previous ones in the literature), which assumes a homogeneous traffic distribution, with a heuristic that adapts STARR-DCS and improves the network performance when the consumption and/or production traffic distributions are not spatially homogeneous. To the best of our knowledge, this is the first work that adapts a DCS network with multiple replicas to heterogeneous traffic conditions.

Paper Organization. The remainder of this paper is structured as follows: Section 2 presents STARR-DCS and describes its operation in detail. The analytical model employed to analyze and optimize resource utilization is described in Section 3. Section 4 compares the performance of random replication versus previous proposals in the literature, and analyzes the benefits and performance of changing replicas over the time. We describe our STARR-DCS implementation and its evaluation in Section 5. A heuristic that could easily adapt operation to scenarios with spatially heterogeneous traffic distributions is introduced in Section 6. Finally, Section 7 offers concluding remarks and discusses the promise of the proposed approach.

2. STARR-DCS OPERATION

We begin by summarizing the main assumptions made in this paper. The focus is on distributed applications operating autonomously over a WSAN without external intervention or communication. The name of an application is known to consumer and producer nodes that participate in the application. The production events and consumption interests associated with a given application are initially assumed to be roughly spatially homogeneous

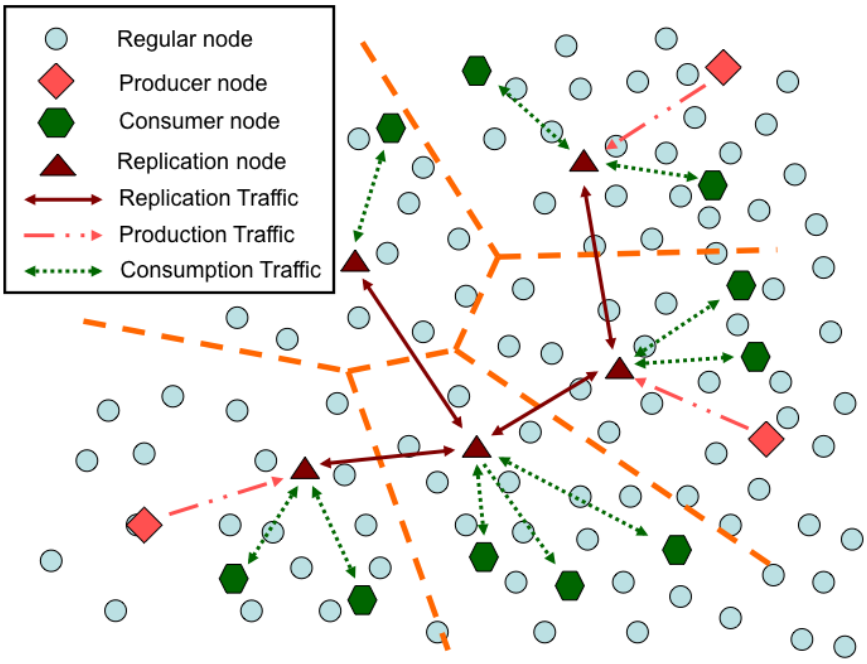


Fig. 1. Example of Data-Centric Storage WSAN with 5 randomly-placed replicas.

(we will relax this assumption in Section 6). We consider a static WSAN 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 as well as the network operational region, and to be able to realize a geographic routing service (e.g. [Karp and Kung 2000]) that can unambiguously route packets to the node which is the closest to a given spatial location.

Below we introduce the functionality required in our proposal for the case where consumption traffic dominates production one, see e.g., Figure 1. Suppose 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 information is known by every node participating in a given application.

Producers and consumers functionality. Suppose a producer (consumer) node generates an event (query) related to APP. Such nodes must first determine the closest replication point by computing the Euclidean distance between their spatial location and that of all replication points obtained from the hash operation: $hash(APP \oplus e \oplus i)$, i=$\{1, 2.., N_r\}$. Once a producer/consumer node determines the closest rendezvous point, it forwards a message/query to that location, i.e., to the replication node r_i closest to its location. In the case of consumption, the rendezvous node just responds with the suitable data to the corresponding query. This replication location will be used for some time, so the producers

and consumers may cache the replication points' coordinates avoiding recomputation for every query/event associated with APP[1].

Creating a tree to replicate data over rendezvous nodes. In the case of production events the next step is creating a *radial spanning tree* [Baccelli and Bordenave 2007] rooted at the closest replication node (e.g. r_1) over which data replication takes place. Each replication node can determine the set of nodes (if any) to which it should forward new data. Since all rendezvous nodes know the hashed locations, we can consider without loss of generality, the construction of the replication tree from the point of view of any given rendezvous node as the root node. The root node, r_1, manages three sets of replication nodes:

— $\mathcal{C}$: the set of rendezvous nodes already covered by the replication tree, where initially $\mathcal{C} = \{r_1\}$ (it contains the root node).
— $\mathcal{R}$: the set of rendezvous nodes to be reached, which initially contains all the rendezvous nodes except the root node: $\mathcal{R} = \{r_2, ..., r_{N_r}\}$.
— $\mathcal{F}$: the set of rendezvous nodes to which the rendezvous node running the algorithm should forward the event, which is initially empty: $\mathcal{F} = \emptyset$.

The algorithm proceeds as follows in the root node. The root node, r_1, computes which rendezvous node in $\mathcal{R}$ is closest to itself. Suppose it is r_2, then r_2 is removed from $\mathcal{R}$ and included in both $\mathcal{C}$ and $\mathcal{F}$., i.e. $\mathcal{C} = \{r_1, r_2\}$, $\mathcal{R} = \{r_3, ..., r_{N_r}\}$, and $\mathcal{F} = \{r_2\}$. Next, it computes the rendezvous node in $\mathcal{R}$ that is closest to any node in $\mathcal{C}$. If the closest distance is between the root node r_1 and r_3, then r_3 is removed from $\mathcal{R}$ and included in $\mathcal{C}$ and $\mathcal{F}$. However, if the closest distance is the one between r_2 and r_3, r_3 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 r_1. Assuming that each node knows who the root is, each one can similarly compute their associated forwarding sets $\mathcal{F}$. Note that if the above distributed mechanism is used, it is possible to obtain a distinct replication tree associated with each rendezvous node serving 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, a single tree could be chosen and shared to distribute events among all replicas.

Alg. 1 exhibits the pseudocode to compute the next hop nodes in the replication tree from the perspective of replication node r_i, assuming a scenario with N_r replication nodes.

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; when a node serving as a replication node (i) exceeds a threshold, E_{th}, on the number of messages sent and received since the epoch started; and, (ii) just before one of such nodes runs out of battery reserves. 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 of the epoch, to the remaining replicas. In turn, based on the exchanged estimates, each replication node

[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.

Algorithm 1 Replication tree construction algorithm run by replication node r_i to know what are the replicas to whom it must forward production events being r_1 the root node.

```
/* Initial sets from r_i */
myself = r_i;
root node = r_1
C = {r_1}
R = {r_2, r_3, ...., r_{N_r}}.
F = ∅
/* Algorithm */
while R ≠ ∅ do
  min_distance = ∞;
  for i = 1 to C.length do
    initial_node = C[i];
    for j = 1 to R.length do
      dest_node = R[j];
      aux_distance = distance(initial_node, dest_node);
      if aux_distance < min_distance then
        min_distance = aux_distance;
        initial_node_selected = initial_node;
        dest_node_selected = dest_node;
      end if
    end for
  end for
  C.add( dest_node_selected);
  R.remove(dest_node_selected);
  if initial_node_selected == myself then
    F.add(dest_node_selected);
  end if
end while
```

computes the minimum estimate for the epoch's residual time based on a common message threshold (E_{th}), along with the number of rendezvous nodes that should be used in the next epoch, based on the overall measured traffic. It must be noted, that these messages containing local traffic measurements must be acknowledged by the other replicas, and in case there is a failure they need to be retransmitted. In addition, before computing the current epoch deadline and number of rendezvous nodes for the next epoch, each replica must ensure that it has received information from all the other replicas (messages containing local traffic measurements) and that its information has been received by all the remaining replicas (acknowledgement). Otherwise, errors in the estimation of the number of replication nodes to be used in the next epoch could easily lead to inconsistencies during the next epoch.

This mechanism to trigger epoch transitions, based on a threshold on the number of messages, can adapt to changing traffic characteristics. Thus, an application could see peak traffic periods in which the selected replication set would use short epochs since it quickly reaches the established message threshold, and for those low-traffic periods where the epoch duration would be much larger since the replication nodes would take a long time to reach the message threshold. Because the application traffic is evaluated once per epoch, the framework can adapt to dynamic applications whose spatial traffic intensities vary over the time.

Finally, when the estimated epoch deadline arrives, current rendezvous nodes know the locations of the current set, and can also compute the locations of the (different) set of

nodes to be used in the next epoch by using the shared epoch-dependent hash function. Now each of the current rendezvous nodes, need only to determine which is the closest node in the subsequent set of replicas (associated locations). If so, such nodes can directly transfer, in parallel, their stored data to the new locations. Such messages would notify the recipients of their (new) role as replicas for the application for the next epoch, so they must be acknowledged.

Consistent notification of epoch changes to producers and consumers. Once the current set of rendezvous nodes decides that an epoch change should be initiated, consumers and producers need to be notified when this change will be executed and the number of new 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, this enables replicas, consumers and producers to experience a smooth epoch transition.

Meta-Information Service. In order to become a viable solution, STARR-DCS has to be further developed to address several practical issues:

— Provide a bootstrapping mechanism for finding the current set of replicas for a given application.
— Provide fault tolerance.
— 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 each of the network applications stores its current epoch value and the number of replication nodes currently in use. Once a new node acquires this information, it can then ask for 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 detailed above. This Meta-Information service is just another application that itself may use the proposed replication framework.

The question now is how a new node is able to know the current epoch of the Meta-Information service. A straightforward solution is just flooding 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 node bootstraps it can simply ask any of its neighbours 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.e. that one coming from the value $i = 1$ in the common hash function) could be the one to notify 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. That is, application's replicas behave as producers of the Meta-Information service. It must be noted that messages notifying such epoch changes must be acknowledged since the information is vital to enable new nodes to participate in the applications. Therefore,

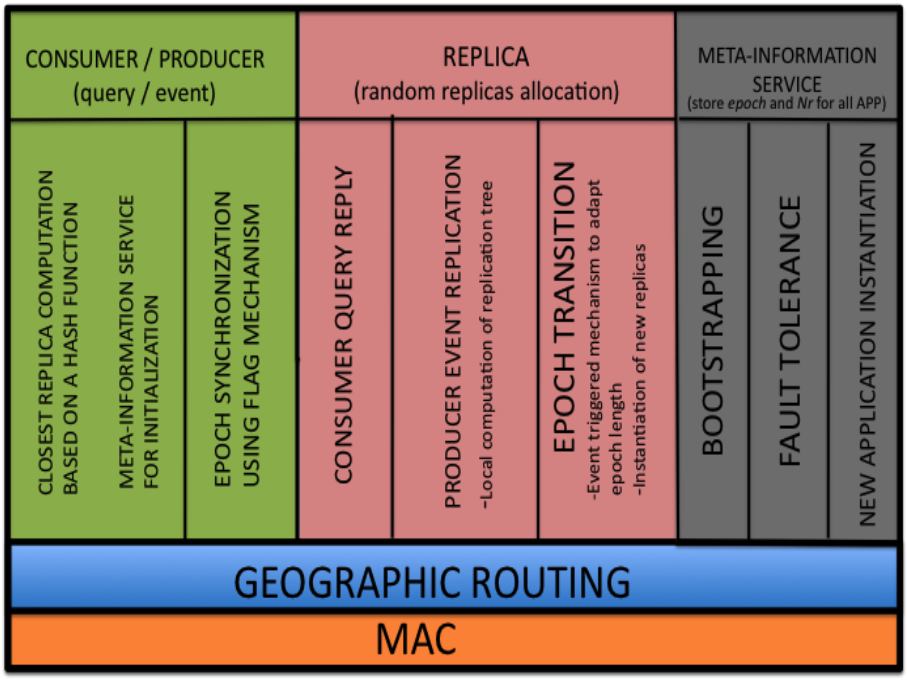


Fig. 2. Potential functionalities performed by a node in the STARR-DCS framework .

if the replication node selected to notify the epoch transition does not receive the acknowledgement, it retransmits it again to the closest Meta-Information service node.

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 still be computed locally by the node. In the case the node 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 STARR-DCS WSAN. When one of the replication nodes of the Meta-Information Service receives a query from a new producer node 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 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 on, any node can start using the new application.

Figure 2 presents a diagram that summarizes all the functionalities described in this section that could be eventually performed by any network node.

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 [Baccelli and Zuyev 1996; Baccelli et al. 1997] and our

own work in applying this methodology to ad hoc wireless networks, e.g., [Baek et al. 2004; Baek and de Veciana 2007].

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 [Stoyan et al. 1995]. 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 and with a small prototype testbed 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.

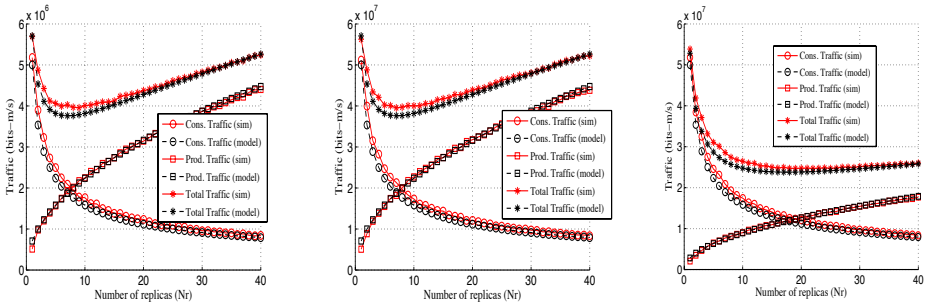
Let us first consider the overall network traffic generated by consumption and production events on the network. The overall metric here is the total traffic load, measured in bits·meter/second that need to be supported by the network, i.e. in region $\mathcal{A}$. Recall that in an ad hoc wireless network traffic load can not simply be measured in terms of bits/s, 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/s 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.

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 [Baccelli et al. 1997] 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 $\frac{1}{2\sqrt{\lambda_r}}$ [Baccelli and Zuyev 1996]. 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.,

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

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* [Baccelli and Bordenave 2007] which includes all the replication nodes. The total length per unit area for radial spanning trees over a homogeneous Poisson Point Process can be computed and is 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



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

Fig. 3. Consumption, production and overall traffic generated by using different number of replication nodes (A=1000x1000 m^2, N=5000 nodes, α=200 bits, β=100 bits) for the case when consumption dominates production

grows as $\frac{\pi x^2 \sqrt{\lambda_r}}{\sqrt{2}}$, so the average length of the tree per unit area is given by $\sqrt{\lambda_r/2}$ [Baccelli and Bordenave 2007]. 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:

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

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:

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

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:

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

Remark 3.1. ***Scaling characteristics.*** Roughly speaking the optimal average number of replicas for the network covering an area A is given by:

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

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/s scales as $\sqrt{A}$ times the

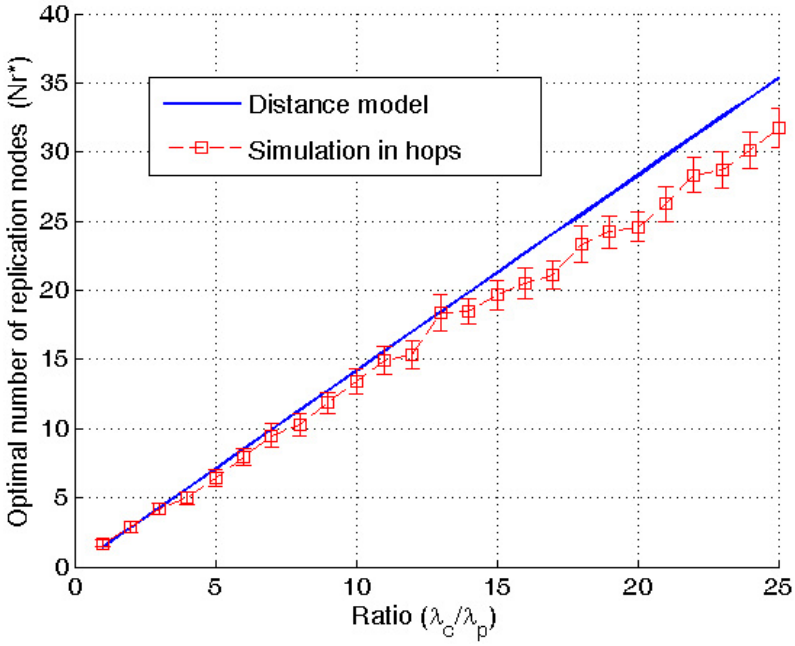


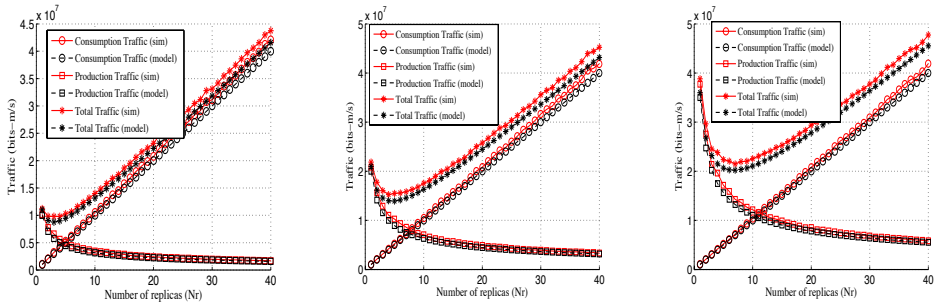
Fig. 4. Optimal number of replicas that minimizes the overall number of messages (A=1000x1000 m^2, N=5000 nodes, Tx=50 m)

geometric mean of the total rate of consumption, $\alpha\lambda_c A$ in bits/s 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 first 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 m^2 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 3 exhibits the overall consumption, production and total traffic measured in bits·m/s obtained by the model and by simulation for three different (λ_c,λ_p) pairs: ($50 \cdot 10^{-6}$, $10 \cdot 10^{-6}$), ($500 \cdot 10^{-6}$, $100 \cdot 10^{-6}$) and ($500 \cdot 10^{-6}$, $40 \cdot 10^{-6}$) $\frac{\text{events}}{\text{s}\cdot\text{m}^2}$. 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 3(a) and 3(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 3(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 multi-hop



(a) λ_c=10·10^{-6};λ_p=200·10^{-6}; N_r^*=2.92 (b) λ_c=10·10^{-6};λ_p=400·10^{-6}; N_r^*=4.64 (c) λ_c=10·10^{-6};λ_p=700·10^{-6}; N_r^*=6.74

Fig. 5. Consumption, production and overall traffic generated by using different number of replication nodes (A=1000x1000 m^2, N=5000 nodes, α=200 bits, β=100 bits) for the case when production dominates consumption.

networks where routes unlikely follow straight paths. 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 WSAN employing greedy forwarding [Karp and Kung 2000] and a transmission range $Tx = 50$ m. We have considered a setup where the ratio λ_c/λ_p varies from 1 to 25.

Figure 4 shows the number of replicas that minimizes the overall simulated traffic based on the actual number of hops of all messages compared to the optimal number of replicas suggested by our model. As 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 information on their queries. 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:

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

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:

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

[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, but such aggregation has implications that are out of the scope of this paper.

The total network traffic is then given by:

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

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

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

The associated minimum overall network cost is:

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

Note that in this regime the optimal intensity of replicas is a more 'complex' function, i.e., cubic of the ratio of production to consumption intensities, yet, in principle, still easily computable by nodes in the field. This model has been also validated via simulation. Figure 5 shows the overall consumption, production and total traffic measured in bits·m/s compared the model and the simulation results for three different (λ_c,λ_p) pairs: $(10\cdot10^{-6},200\cdot10^{-6})$, $(10\cdot10^{-6},400\cdot10^{-6})$ and $(10\cdot10^{-6},700\cdot10^{-6})$ $\frac{\text{events}}{\text{s·m}^2}$. The number of replicas employed varies from 1 to 40. Thus, the optimal number of replicas for these cases is 2.92, 4.64 and 6.74 respectively. As seen in the figure the model is very close to the simulation results.

3.2. Evaluating storage limits

If multiple applications share the same network storage resources, say a storage capacity of b bits per node, this may limit the amount of replication one can use. To better understand this, consider a network where m homogeneous applications, i.e., with the *same* consumption and production intensity and data storage requirements, say d, share a network with an intensity of λ_n nodes/unit area in region $\mathcal{A}$.

To model memory utilization in replication nodes, suppose a given application selects the nodes to serve as replication nodes as follows. It generates random spatial locations Π_r with intensity λ_r on the plane, and then network nodes that are the closest ones to these locations are chosen as replication nodes. Note if several points in Π_r are close to the same node, then that node is used only once. Let V be a random variable denoting the area of the Voronoi cell of a typical network node. If at least one point in Π_r is in the Voronoi cell of such node, it is selected as a replica. The probability that the region with area $V = \mathcal{V}$ contains no point from the process a Π_r locations, is given by its void probability $p(\mathcal{V}) = e^{-\lambda_r\mathcal{V}}$[Stoyan et al. 1995]. So the average probability a typical node is chosen by an application using a intensity λ_r for choosing replication nodes is given by:

$$1 - E[p(V)] \;=\; 1 - E[e^{-\lambda_r V}] \approx \lambda_r E[V] - \frac{\lambda_r^2}{2}E[V^2] = \frac{\lambda_r}{\lambda_n} - 0.62\frac{\lambda_r^2}{\lambda_n^2}$$

where we have used the fact that $E[V] = \frac{1}{\lambda_n}$ and also that $\sqrt{\text{Var}(V)} = E[V](0.52)$ [Moller 1994].

Let X_i be a Bernoulli random variable which is 1 if application i uses the node as a replication site, and zero otherwise, i.e.,

$$P(X_i = 1) = 1 - E[p(V)] \;\text{ and }\; P(X_i = 0) = E[p(V)].$$

Suppose a given node has enough storage for b/d different application's data, then the probability that it is overloaded is given by:

$$P(\sum_{i=1}^{m} X_i > b/d).$$

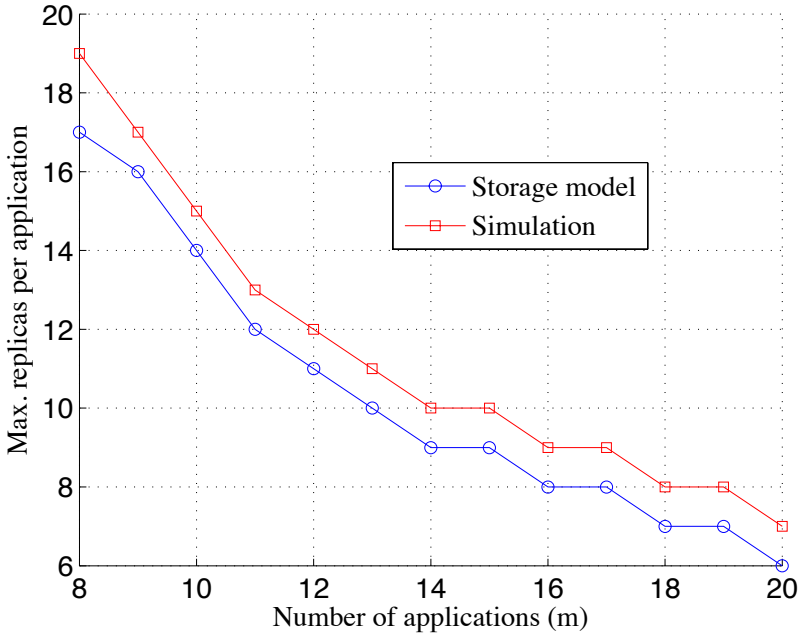


Fig. 6. Maximum number of replicas per application to keep the probability of node saturation below a 10% (A=1000x1000, N=100, b/d=3, δ=0.1).

Note that X_i are not independent, because if a cell has a larger area, they are more likely to be 1. In other words they are only conditionally independent given the area of the cell. To estimate the overload probability, we shall still approximate the above sum as a Gaussian random variable, i.e., $\sum_{i=1}^{m} X_i \sim N(\mu, \sigma^2)$ where μ and σ^2 correspond to the mean and variance of the sum. In particular, as shown above:

$$\mu = E[\sum_{i=1}^{m} X_i] \approx m\frac{\lambda_r}{\lambda_n} - 0.62\frac{\lambda_r^2}{\lambda_n^2}.$$

To compute the variance of the sum we can condition on the size of the cell V to obtain:

$$\begin{aligned}\sigma^2 &= \mathrm{Var}(\sum_{i=1}^{m} X_i) = E[\mathrm{Var}(\sum_{i=1}^{m} X_i|V)] + \mathrm{Var}(E[\sum_{i=1}^{m} X_i|V]) \\ &= E[mp(V)(1-p(V)] + \mathrm{Var}(m(1-p(V))) \\ &= mE[p(V)(1-p(V)] + m^2(E[(1-p(V))^2] - E[1-p(V)]^2).\end{aligned}$$

Further expanding the terms in the previous equation, we obtain:

$$\sigma^2 \approx m\left(\frac{\lambda_r}{\lambda_n} - 1.9\frac{\lambda_r^2}{\lambda_n^2}\right) + m^2\left(0.27\frac{\lambda_r^2}{\lambda_n^2} + 1.27\frac{\lambda_r^3}{\lambda_n^3} - 1.61\frac{\lambda_r^4}{\lambda_n^4}\right).$$

Now given these results we can roughly assure that the risk of running out of storage space for a typical sensor is less than δ by requiring that:

$$P(\sum_{i=1}^{m} X_i > \frac{b}{d}) \approx Q(\frac{\frac{b}{d} - \mu}{\sigma}) \leq \delta$$

where $Q()$ denotes the complementary distribution function of a standard Gaussian random variable. This in turn gives a requirement that

$$\frac{b}{d} \geq \mu + t(\delta)\sigma$$

where $t(\delta)$ is such that $Q(t(\delta)) = \delta$.

This can be interpreted as a constraint on the maximum number of homogeneous applications one can support, or the maximum replication rate per application one can allow.

In order to validate the model, we simulated a network where the requirements on nodes' storage were fairly high. This is the case where the Gaussian approximation is effective and the model can provide useful results for network designers. Specifically we have simulated a network with $N = 100$ nodes, and varied the number of applications from 8 to 20, and the number of replicas per application from 1 to 20. We considered the case where $b/d = 3$, i.e., a node can simultaneously support at most 3 applications. For each scenario we evaluated the maximum number of replicas each application could use while ensuring that a typical node's saturation probability was lower than $\delta = 0.1$ both via simulation and with our analytical model. Figure 6 shows that the storage model and the simulation results are very close, showing a difference of just 1 replica in most of the cases. Moreover, it must be noted that the model is conservative, since it provides a lower value than the simulation, which is desirable for safe network design.

The importance of these results is as follows. When multiple applications share the network infrastructure, our analysis shows that depending on the production and consumption intensity they may choose to use a large number of replicas. However in doing so, it may require replicas to store more data than they are in fact capable of. So in practice the intensity of replication associated with multiple information services sharing the network may need to be limited, to preclude this overload from happening.

3.3. 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 does not need to have the same intensity as the current one. Also suppose each replica node currently holds an average amount of data s.

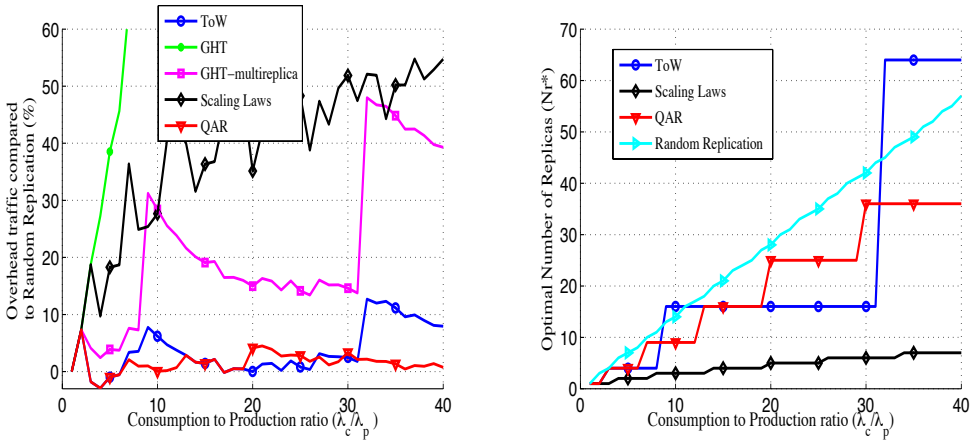
A rough estimate of the energy cost associated with moving data from the current set of replication nodes to the new one $T_r(\lambda_r^c, \lambda_r^n)$, can be evaluated as follows. Each old replica would contact one of the new nodes. Given that the distance to a new randomly located replica from one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

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

So if $\lambda_r^n = \lambda_r^c$ the cost is $T_r = \frac{s}{2}\sqrt{\lambda_r}$ 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



(a) Overhead traffic of other solutions when compared to Random

(b) Number of replicas used for the different proposals: ToW, QAR, Scaling-Laws and Random

Fig. 7. Random vs. ToW, QAR, Scaling-Laws, GHT and GHT with multiple replicas (A=1000x1000 m^2, N=5000 nodes, Tx=50 m, α=200 bits, β=100 bits)

.

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

4.1. Random vs. grid-based and uniform replica allocation

4.1.1. Quantitative Comparison. We have compared random replication that is the replication mechanism used in the proposed STARR-DCS framework with those that are similar in spirit: ToW [Joung and Huang 2008], QAR [Cuevas et al. 2010], Scaling Laws [Ahn and Krishnamachari 2006], the original GHT proposal [Shenker et al. 2003], which uses a single replication node, and GHT with multiple replication nodes [Ratnasamy et al. 2002; Ratnasamy et al. 2003]. 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 use the same 4^d geometric formula for the number of rendezvous nodes.

In order to compare these approaches, we ran simulations for a large WSAN with the following characteristics: an area $A = 1000 \times 1000$ m^2, $N = 5000$ nodes, transmission range $Tx = 50$ m and λ_c/λ_p traffic ratio 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 by all messages as the measure of the overall traffic cost.

Figure 7(a) shows the network traffic improvement achieved using random replication compared to all the other approaches, and Figure 7(b) 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 it allows a finer granularity at which the number of replicas being used can be fine tuned. 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 7(b)). 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.

We conclude that random replication is the approach best minimizing the overall network traffic improving over all previous approaches in the literature that use grid-structured or uniform replication. Moreover, processing the random locations for the rendezvous nodes is easier for different sensornet shapes than those using structured replication. Hence, random replication is simpler and more cost-effective. Therefore, all these results validate our option of choosing random replication as the replication mechanism in the proposed solution STARR-DCS.

4.1.2. Qualitative Comparison. It must be noted that the use of randomly-located replication nodes can be used whatever the underlying routing protocol is as long as it is able to identify routing addresses for replication nodes. However, structured replication mechanisms such us ToW [Joung and Huang 2008], QAR [Cuevas et al. 2010], Scaling-Laws [Ahn and Krishnamachari 2006] and GHT with multiple replicas [Ratnasamy et al. 2002], can be only applied when geographic routing is used. Therefore, if we think of a distance vector or link-sate routing protocol based on node-ID in which a node knows at least the next hop towards a given destination, it is simple to select a random set of replication nodes. However, by just knowing the nodes ID, it is not possible to allocate the replication nodes in a grid fashion because the nodes do not know their own location information, and therefore the set of nodes creating a grid replication structure cannot be found.

Therefore, the only approach that works with an important number of ad-hoc routing protocols (e.g. pathDCS [Ee et al. 2006]) is random replication.

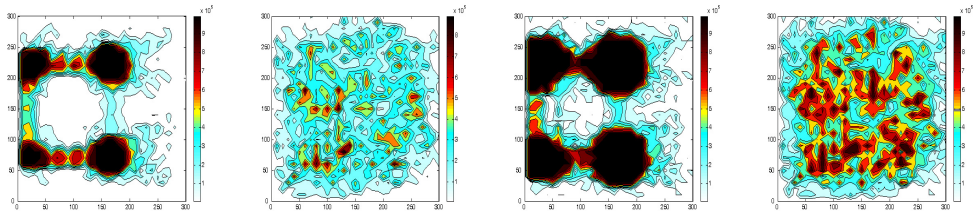
4.2. Changing Replicas over the time

Nodes 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 first. In [Shenker et al. 2003], [Ratnasamy et al. 2002], [Ratnasamy et al. 2003], [Cuevas et al. 2010] and [Joung and Huang 2008], when this happens, an alternate node 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 whole network.

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 to avoid routing holes, 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 of rendezvous nodes over time.

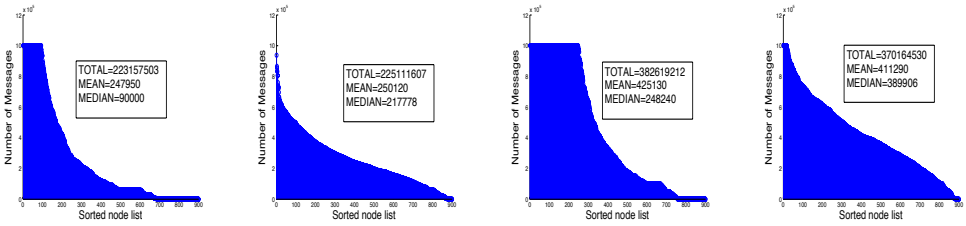
In order to verify the abovementioned statements, we ran simulations comparing ToW [Joung and Huang 2008] that uses static replicas (ToW-static) with STARR-DCS, where the set of replication nodes changes over the time.

We use a grid-based node deployment (which makes energy maps generation easier) with $N = 900$ nodes, over a square of area $A = 300 \times 300\,\mathrm{m}^2$. Each node has a transmission range $Tx = 30$ m. In addition, it must be noted that we evaluate the case where consumption dominates production ($\lambda_c > \lambda_p$). We use the number of messages in the network as a first order proxy for consumed energy. A sensor node's energy is depleted once it sends and/or



(a) ToW-Static after 30K cycles (b) STARR-DCS after 30K cycles (c) ToW-Static after 50K cycles (d) STARR-DCS after 50K cycles

Fig. 8. Energy map from the number of messages sent and received by all nodes of the network (A=300x300 m^2, N=900 nodes, Tx=30 m, N_p=100 producers, N_c=300 consumers, L=10 cycles, M=1000 $\frac{\text{messages}}{\text{epoch transition}}$ change, Battery = 10^6 messages, E_{th}=3*10^5 messages).



(a) ToW-Static after 30K cycles (b) STARR-DCS after 30K cycles (c) ToW-Static after 50K cycles (d) STARR-DCS after 50K cycles

Fig. 9. Distribution of the number of messages per node (A=300x300 m^2, N=900 nodes, Tx=30 m, N_p=100 producers, N_c=300 consumers, L=10 cycles, M=1000, $\frac{\text{messages}}{\text{epoch transition}}$, Battery = 10^6 messages, E_{th}=3*10^5 messages).

receives one million messages. Finally, the threshold that determines the end of an epoch, E_{th}, is set to 300000 messages (30% of the battery) [3].

For these simulations we have used geographical routing based on greedy forwarding [Karp and Kung 2000]. When greedy forwarding fails, e.g. due to routing holes, we use the shortest path from the node where the greedy forwarding stopped to the destination node.[4]

Time is measured in cycles in order to scale the simulations and be able to deploy a larger number of nodes. A cycle is the 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*. 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 STARR-DCS.

In order to measure the cost of an epoch change, we assume that the produced data has a mean lifetime of L cycles. Then, the average data at each replication node is $d = N_pL$ 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

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

[4]We do not implement the face routing facility of GPSR due to its complexity. We used a shortest path approach to overpass the routing holes instead. Both of them lead to very similar paths in our simulation. Thus we meet our goal of comparing the effect of changing replication nodes position over the time instead of keeping them static.

Table I. WSAN Lifetime ToW-Static vs STARR-DCS

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
STARR-DCS (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%

new ones. By having 100 producers, this means a total cost of M=1000 messages per epoch change.

Figure 8 shows the energy distribution map after a simulation time of 30000 and 50000 cycles. Figure 9 shows the number of messages sent and received by each node at the same cycles, as well as the mean and median values per node 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 8(a) and 8(c), keeping static replication points creates routing holes in the network, with 93 and 247 expired nodes after 30000 and 50000 cycles respectively. The number of depleted nodes are only 0 and 17, respectively, when replication nodes are changed over the time, that is, when STARR-DCS is in place. Furthermore, simulation results obtained later in time (70959 cycles) show that ToW-static network is eventually disconnected, because holes become very large and coalesce. In addition, more nodes participate in the network operation when STARR-DCS is used. As shown in Figure 9, all nodes except 18 (2%) after 30000 cycles and 15 (1.7%) after 50000 cycles, have sent and/or received at least one message, whereas in the case of ToW-static more than 200 (22%) 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, STARR-DCS 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 I, we compare the network lifetime using both approaches: ToW-static and STARR-DCS. Since lifetime can be defined using different metrics [Dietrich and Dressler 2009] (first node running out of battery, some percentage of nodes 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 10 shows how using a message threshold to trigger epoch changes compares to employing a fixed epoch duration as proposed in [Thang et al. 2006] (note that this work actually refers to a single rendezvous node scenario and it proposes to change motivated by storage saturation instead of energy issues). We simulate a larger (N=5000 nodes, A=1000 $\times$ 1000 m^2, Tx=50 m) multi-application WSAN with random replication. We set up m=5 heterogeneous applications with (20, 40), (60, 120), (100, 200), (140, 280) and (180, 360), ($\frac{\text{production-events}}{\text{cycle}}$, $\frac{\text{consumption-queries}}{\text{cycle}}$) pairs, calculating the network's lifetime (1% of nodes expire) for both the two dynamic approaches versus using a fixed static set of randomly located replicas. Again the need for 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. That is, the set of values providing good values

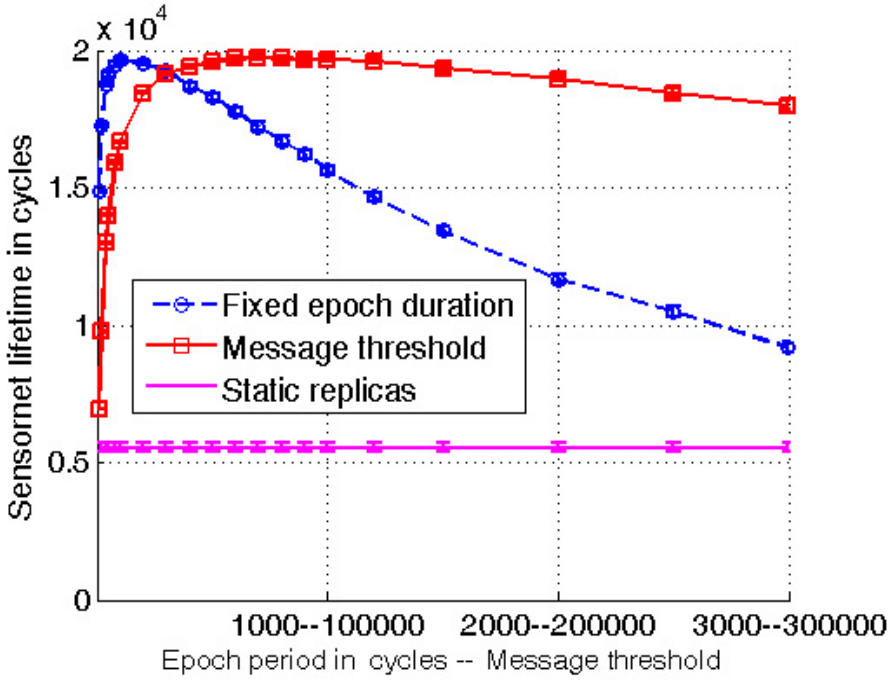


Fig. 10. WSAN lifetime comparison (A=1000x1000 m^2, N=5000 nodes, Tx=50 m, L=10 cycles, m=5 applications, Battery=10^6 messages). X axis refers to the cycles for changing the epoch in the fixed duration approach, or the message threshold.

of network lifetime represents a very small window when employing fixed epoch duration whereas our solution shows a larger window to choose a message threshold value leading to a good network lifetime.

4.3. Epoch duration analysis

We have already demonstrated the great improvement of changing the replication nodes over the time. Thus the next question is whether it is better to use shorter or longer periods before an epoch transition. At first glance, the best solution seems to be using short periods so that the load is better spread among the nodes. However, as we have already seen, there are some overheads associated with epoch transitions, such as moving all the data stored in the current replicas to new ones. Considering this trade-off, using shorter epoch periods will lead to balance the energy consumption among the nodes, thus reducing the energy consumption variance per node, but it would also increase the average energy expenditures per node, which means increasing the overall energy expenditure. By contrast, using longer epoch periods increases the variance of thexww node energy consumption since nodes will keep being replicas for more time, but it will reduce the overall (average) traffic on the network.

Thus the key of this trade-off is determining how much extra energy should be spent to balance the energy among the nodes. The decision depends on each particular application. For instance, for an application where all the nodes are needed, so that all of them should kept alive together in order to allow the application to work properly, (i.e. extend the time when the first node runs out of battery) the right selection is to balance the energy as much as possible, by means of using short epochs (frequent epoch changes). Of course the price of doing so is that a lot of extra energy is required for those frequent epoch changes. Thus, if

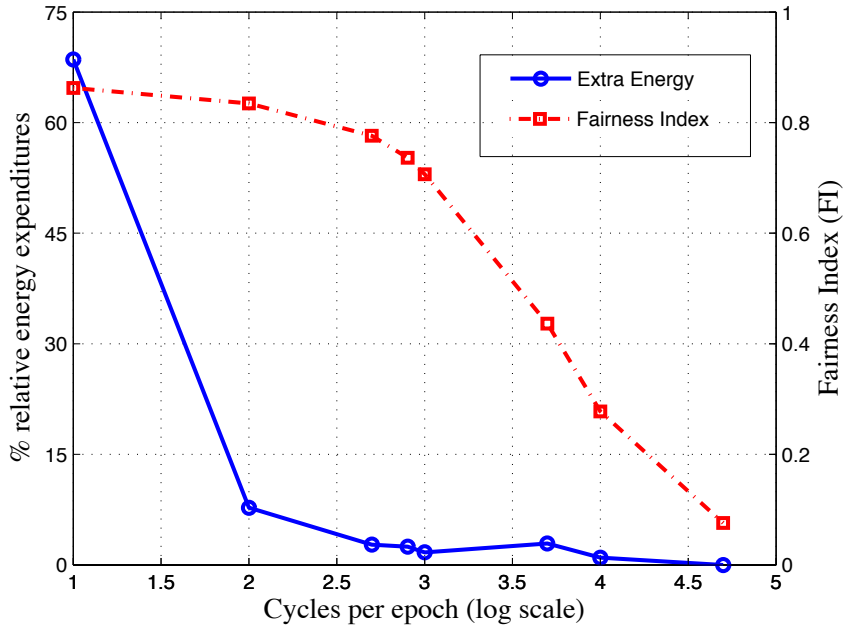


Fig. 11. Epoch duration analysis (A=300x300 m^2, N=900 nodes, Tx=30 m, L=10 cycles, Simulation Time=50000 cycles).

the application requirement is to reduce the overall energy consumption, then very frequent changes would be a poor choice.

To evaluate this trade-off we have run simulations in a WSAN with the same simulation parameters used in the previous section. That is, a grid deployment in an area A=300x300 m^2, where N_s= 900 nodes. N_c=300 consumers and N_p=100 producers are supported with a transmission range T_x=30 m. Ten messages per producer are stored in the replicas (L=10 cycles and each producer generates 1 message per cycle). It must be noted that L, along with the production rate, are the factors that establish how costly an epoch transition is in terms of energy. Finally, we use the number of messages sent and received by each node as an approximation of the energy consumed by the network.

We evaluated the network performance using different fixed epoch durations: 10, 100, 200, 500, 1000, 5000, 10000 and 50000 cycles per epoch. For each case we ran the simulation for 50000 cycles, resulting in a range of 5000 epoch changes to none. For each epoch duration we ran 50 simulations over which we averaged the performance. Note, that nodes do not run out of energy in this experiment, so no routing holes are generated, and that is why in terms of traffic overhead, the best option is to have no changes. However, as it was previously demonstrated the need for longer routing paths could have a larger impact on traffic overhead than the change of replication nodes over the time.

We use the Fairness Index (FI) [Jain et al. 1984] as a measure of how well balanced the energy consumption is across the nodes for different epoch durations. The FI goes from 0 (lowest fairness) to 1 (highest fairness) and it is defined as,

$$FI(x_1, x_2, x_3, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$$

where x_i represents the number of messages sent and received by node i. In addition, we measure the relative energy expended in the network for different epoch durations (more or less changes during the simulation time) compared to the minimum energy case (i.e. no changes). Figure 11 shows the FI and the relative energy consumed for different epoch durations on a logarithmic scale.

As we could expect, the lower the number of epochs per cycle (the more epoch changes) the better the network fairness, but the greater the overall energy required. However, there is a region, around 500 cycles per epoch (between 2.5 and 3 in the x-axis in the figure), where the extra energy consumed is not that big, below a 5%, and the FI is very good (>0.75). This operational regime heavily depends on the value of L, which impacts the overhead associated with epoch changes. If this cost is low this region moves to the left, resulting in a better FI and a lower energy requirement. However, if the transition cost is very high, the region with a low energy demand (compared to the best case) will move to the right, of course producing worse FI values, thus a bigger variance for the energy consumed per node. Although the abovementioned region could be a good operational regime area in general, we note that each application may have a different suitable operational regime.

5. STARR-DCS IMPLEMENTATION

We have implemented STARR-DCS on real motes. Our implementation supports an arbitrary number of replicas that change over the time, and uses the Meta-Information service for bootstrapping purposes. Furthermore, replicas are able to exchange traffic measurements and compute the remaining time of the current epoch, as well as the number of replicas for the next epoch. These provide an efficient synchronization mechanism for all the entities involved in a particular application: consumers, producers and replication nodes.

To test the implementation we have used 20 Jennic motes of two different types: JN-5121 (5x) and JN-5139 (15x) . The JN-5139 wireless microcontroller device integrates a 32-bit RISC processor, with a fully compliant 2.4GHz IEEE 802.15.4 transceiver, 192kB of ROM, 96kB of RAM, and several analogue and digital peripherals. The JN-5121 is an older version and it only has 64kB of ROM.

5.1. Implementation details

A node can initially be assigned one of these three roles: producer, consumer or relay (in which case it is neither a consumer nor a producer). The role of a particular node is expected to be specified by the application using the framework.

Producers generate events via two different mechanisms: (i) at a predefined rate, e.g. temperature reading every minute; (ii) manually when a button is pressed. Moreover, the number of data elements per event can be configured by the application (e.g. three temperature samples per production event). Consumers also generate queries using the same mechanisms utilized by the producers: at a constant query rate or triggered by a pressed button.

We implemented a greedy forwarding routing on the motes as the routing layer to be used by our framework. In order to avoid more complex routing operations (e.g. face routing), we set up scenarios in which it was feasible to route a message from any source to any destination node by only using greedy forwarding. In these scenarios, if a node receives a message and it is closer to the destination coordinates than any of its neighbours, then it is the closest one to the destination coordinates. The mechanism used to choose the nodes acting as replicas is the closest one to a randomly selected spatial location.

We have defined a common header to be used by all the protocol messages required to implement the STARR-DCS framework. Figure 12 shows all the different fields included in the header. Next, we describe each of these fields:

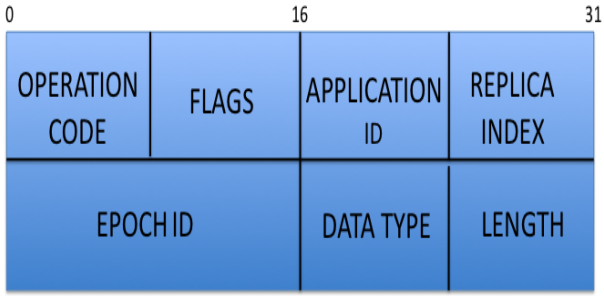


Fig. 12. Protocol Header to implement STARR-DCS.

— **OPERATION CODE:** Defines the type of message (e.g., PUT, GET, GET REPLY, etc).
— **FLAGS:** Is used for special operations, i.e. consumers and producers use a flag to indicate that they do not know yet when the current epoch finishes and what is the number of replicas to be used in the next epoch. This field is also used to request an acknowledgement.
— **APPLICATION ID:** Defines the application that is using the framework.
— **DATA TYPE:** Is used to define the data structure employed by the application,
— **LENGTH:** Indicates how many data structures are included in the message.
— **REPLICA INDEX:** Identifies (in the case that is required) the replication node that is the source of the message (e.g., that information is needed in all the replication nodes in order to generate the replication tree in a distributed fashion).
— **EPOCH ID**: Specifies what is the current epoch of the source node and it is used for synchronization purposes.

Next, we present all the different message types required in our implementation that are identified by different OPERATION CODE values:

— **PUT**: Is the message used by producers to send the measured data to the closest replication node. In addition, it is also used for updating the epoch and number of replicas of a given application stored in the Meta-Information service. Then, once the replicas have obtained the epoch expiration time and the number of replicas in the next epoch, the first one (assigned the ID 1 in the hash operation) sends a PUT message to the Meta-Information service node, indicating the next epoch ID, the number of replicas in the next epoch and the time in seconds until the current epoch expires (similarly as it is done with consumers and producers). Therefore, the Meta-Information service node sets up a timer that ends at the end of the current epoch, until that moment it still serves the information for the current epoch, after it will start serving the information of the next epoch. When the PUT message is used to update the Meta-Information service information a flag is used to indicate that an acknowledgement is required.
— **PUT ACK**: Is the message used to acknowledge a PUT message that requires a confirmation.
— **EPOCH ADVERTISEMENT**: This message is used to notify producers about the epoch duration and the number of replicas to be used in the next epoch. Then, this message contains the number of replication nodes that will be deployed in the next epoch and the time in seconds until the end of the current epoch.
— **GET**: Is the query message used by consumers to obtain information from the closest replica. GET messages are also used when a new producer or consumer contacts the Meta-Information service in order to obtain the current epoch and number of replication nodes of the application it wants to participate.

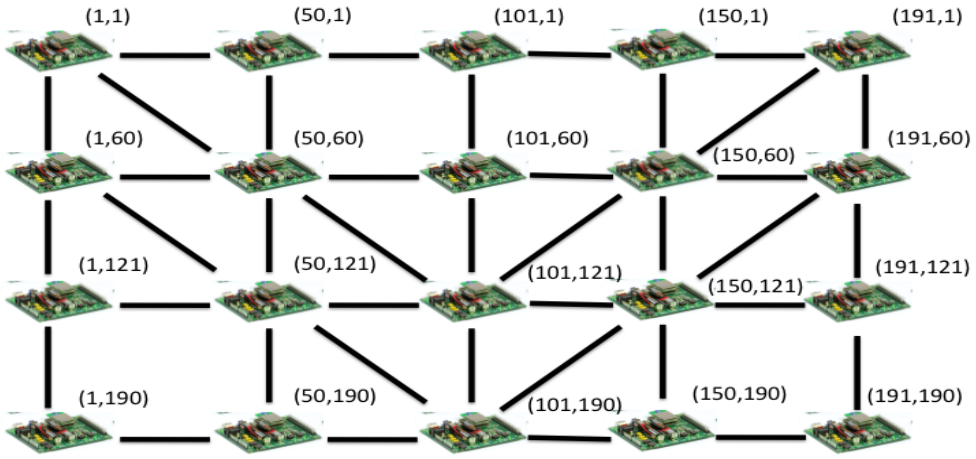


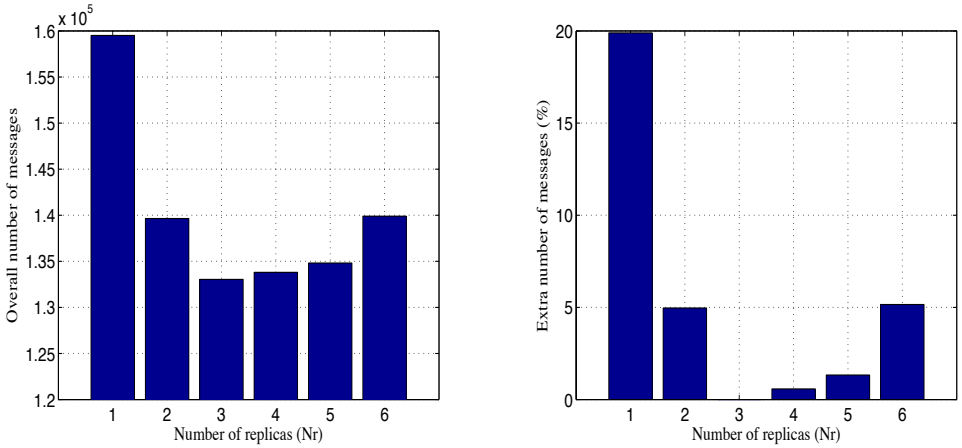
Fig. 13. Real testbed using 20 motes emulating a 200x200 square meter network in a two dimensions plane

.

— **GET REPLY**: Is the message that answers consumers' (or producers' when using the Meta-Information service) GET request. It contains the suitable application information requested by the consumer (or producer). In addition, the information about the epoch duration and the number of replicas to be used in the next epoch is piggybacked in this message for consumers. .
— **REPLICATION**: Is the message used to replicate the production data received by one replica in the remaining replicas.
— **REPLICATION STATISTICS**: Is a message used by a replica to send other ones the consumption and production traffic accounted during the measurement period. This message is sent through the replication tree and it is retransmitted after a pre-defined time if some acknowledgement is not received.
— **REPLICATION STATISTICS ACK**: Each replica receiving a REPLICATION STATISTIC message sends an ACK back to the source node in order to notify that it received the measurement information. This message does not use the replication tree but it is sent using a direct path.
— **REPLICA INSTANTIATION**: When the epoch expires, old replicas send a REPLICA INSTANTIATION message to the suitable new replication nodes, which become replicas for the new epoch after receiving this message.
— **REPLICA INSTANTIATION ACK**: New replicas acknowledge the ones in the previous epoch that they have been instantiated and they are performing the replica role in the new epoch.

It must be noted that we have implemented the Meta-Information service in a single static node and we have checked that the bootstrapping and fault tolerance services work fine. Therefore, nodes initialized after the application has run during several epochs are able to synchronize and normally produce or consume data over the time.

Moreover, our implementation covers a multi-application environment. In particular, we successfully ran 4 different applications in parallel on our testbed.



(a) Overall number of messages using different number of replicas in the network.

(b) Extra percentage of messages with respect to the best value for the number of replication nodes (N_r= 3).

Fig. 14. Evaluation of the optimal number of replicas

5.2. Network evaluation

We consider three different aspects in our evaluation: (i) we have checked that the optimal number of replicas provided in Eq. 1 is useful in a real testbed, (ii) we have checked that changing the optimal number of replicas over the time balances the energy consumption, and (iii) we have checked that STARR-DCS extends the WSAN lifetime.

In all cases we used a scenario with 20 nodes located in 4 rows by 5 columns grid fashion emulating a 200x200 square meters network. Each node was programmed with its own coordinates and its neighbours coordinates. We did not use a full mesh connectivity, but a more irregular one in order to create longer communications paths within the network.

Figure 13 shows the testbed used to evaluate our framework implementation. It must be noted that our framework is focused on large networks, for instance our network traffic model assumes an infinite plane, but in this real scenario we are working with a small network with just 20 nodes.

Our performance metric was the overall number of messages sent and received within the network[5]. We demonstrate in this section that the main results obtained from the simulations of large WSANs are also valid in a real testbed. In addition, the fact that we are using a small network reinforce even more the paper outcomes.

5.2.1. Optimal Number of Replicas. In this test we always use the same number of replicas during the experiment, irrespectively of the traffic statistics collected by the replication nodes. We tested different scenarios using from 1 to 6 replication nodes. We measure the total number of messages (sent and received) to account the traffic generated in the network, which is also valid as a rough estimation of the network's energy expenditure. We use a single application in the test, which was run for 3 hours.

All the nodes were producers and consumers at the same time, however the consumption and production rates were different. The nodes generate a production event every 45 seconds, while they generate a query every 15 seconds. Following, the formula of Eq. 1, the optimal number of replicas would be N_r^*=4.2. Figure 14(a) shows the overall number of messages

[5]Jennic 5139 and 5121 motes do not allow to measure the remaining battery in the mote, so we could not use energy to directly analyze the network performance

in the network when forcing the framework to use 1, 2, 3, 4, 5 and 6 replicas. The number of replicas that minimizes the overall traffic is 3.

Figure 14(b) shows the extra traffic generated when using a number of replicas different than 3. Then if 4 replicas are selected, which is the value chosen by our model, only a 0.5% extra traffic is generated. The other close value to that obtained from the model is 5 replicas. In this case, the overhead traffic is an 1%. Finally, if we choose those values far from the one provided by our model the extra traffic grows up to a 5% for 2 and 6 replication nodes and up to a 20% when a single replication node is selected. This demonstrates in a real testbed that always using a single replication node as proposed in the seminal DCS work [Shenker et al. 2003] could generate a lot of extra traffic.

This test demonstrates that even if we are far from the model assumptions (i.e. infinite field, distance-based model instead of hop-based model, etc) the optimal number of replicas that the model provides is leading to good results in terms of minimizing the network traffic (thus to reduce the network energy consumption) eliminating the uncertainty of what number of replication nodes should be used.

5.2.2. Balancing Energy Consumption. The second test we have conducted is to verify whether changing the replication nodes over time balances energy consumption per node even though it generates extra messages associated to epoch transitions. For that we compare a static scenario with two dynamic ones implementing STARR-DCS that were configured with different message thresholds to change the epoch. We remember that the message threshold is the maximum number of messages that a node can send and receive while playing the role of replication node. In addition, it must be noted that in this test none node ran out of battery since we have a small testbed with only 20 nodes, and thus no network holes appear in the network under the static scenario as it happened in the simulation experiments (see Section 4). We remember that the appearance of routing holes in the static scenario leads to a larger traffic overhead than the one related to epoch transitions in the dynamic scenario. However, in the current test the dynamic solution presents a higher overhead due to the absence of routing holes in the static scenario.

The test ran for 3 hours and all the nodes were consumers and producers at the same time. The production rate was 1 message every 45 seconds and the consumption rate was 1 query every 15 seconds. As in the previous example this leads to an optimal number of replicas of N_r^*=4.2, so we decided to use 4 replicas in the static case, whereas the STARR-DCS tests also started with 4 replicas. After the first epoch, the STARR-DCS framework computed the number of replicas in the next epoch based on the traffic statistics captured by the replicas during the measurement period that lasts 1 minute. It must be noted that usually the selected number of replicas was 4, but sometimes one node was taking the responsibility of two hashed locations, therefore the real number of replicas was actually 3. Finally, we have used two different values for the message threshold in order to manage shorter and longer epochs. Those values were 240 and 360 messages respectively. We remember that shorter epochs are expected to provide a better fairness at the price of a higher overhead since more messages are generated due to more frequent epoch transitions.

Hence, in order to compute the network fairness we use the Fairness Index (FI) [Jain et al. 1984] over the number of messages sent and received by each node. In addition, we also account the extra number of messages generated by the dynamic scenarios in comparison with the static case.

Table II shows the expected results. On one hand, changing the replication set over the time leads to a much fairer energy distribution in the network than using static replicas. In addition, as we expect the shorter the epoch (a lower message threshold) the better the energy distribution. That is the reason why the scenario with a 240 message threshold presents a better FI than the one with a message threshold of 360. On the other hand, changing the replication nodes over the time leads to increase traffic overheads, that in our

Table II. Evaluation of energy distribution. Fairness Index (FI) of messages sent and received by network nodes and overhead generated by STARR-DCS compared to a static solution using random replication

	Static	**STARR-DCS 240**	**STARR-DCS 360**
FI	0.59	0.83	0.81
Overhead (%)	0.00	10.53	7.04

test was a 10% for the STARR-DCS test with shorter epoch (240 messages threshold) and a 7% in case of using a longer epoch (360 messages threshold).

Moreover, we note that we have used a low message threshold to generate quite a few epoch changes so as to better see the distribution of the energy consumption within the network. Then, using a 240 message threshold in a 3 hours testbed means 30 epochs (i.e. 9-10 changes per hour), while a 360 message threshold leads to 20 epochs (i.e. 6-7 epochs per hour).

Therefore we have confirmed in our testbed that changing the set of replication nodes over the time leads to a much fairer energy consumption distribution within the network.

5.2.3. First node dead lifetime. Due to the reduced number of nodes we were using and the fact that Jennic 5139 and 5121 motes do not allow to monitor the remaining battery in a node, we decided to emulate the node battery lifetime with a limit to the maximum number of messages sent and received by the nodes.

For that purpose we evaluated: a static scenario with random replicas, and two scenarios implementing STARR-DCS with message threshold values of 240 and 360 messages, respectively. For each of these scenarios we measured the time when the first node reaches 5000, 6000, 7000, 8000, 9000 and 10000 messages in order to demonstrate that changing the replication set over the time allows an effective lifetime extension.

Table III. Evaluation of lifetime extension when adopting the STARR-DCS solution instead of the static one. We obtain the time when the first node reaches different thresholds

Lifetime (# msg.)	**Static (sec)**	**STARR-DCS 240 (sec)**	**STARR-DCS 360 (sec)**	**STARR-DCS 240 improv. (%)**	**STARR-DCS 360 improv. (%)**
5000	3054	4199	3506	37%	15%
6000	3659	5142	4556	41%	25%
7000	4242	6268	5456	48%	29%
8000	4830	7203	6301	49%	30%
9000	5419	8362	7187	54%	33%
10000	6003	8945	7863	49%	31%

Table III shows the effective time extension in percentage of the STARR-DCS solution compared to a static one. The first conclusion is that changing the replication set over the time reduces the charge of the most saturated node in the network. This is translated into a longer period to reach a given number of messages, which implies a longer time before running out of battery. In particular, when analyzing the scenario implementing STARR-DCS with an epoch message threshold of 240, we check that in all the cases the time extension is over a 35%, and even overpasses a 50% when the messages limit is established in 9000 messages. The time extension for an epoch message threshold of 360 is reduced to values between a 15% and a 33% depending on the different messages limits. This is happening because a lower message threshold implies a shorter epoch, thus a better distribution of energy per node. This reduces the number of messages in the node with more messages at the price of increasing the overhead in the overall network.

Finally, we note that the trend in the results is that the higher the limit of messages, the longer the time difference between the static and the framework solutions. That means that if we were able to run very long tests (e.g. 1000 epochs) like the ones evaluated in the

simulations (see Section 4) the lifetime extension would be much higher, as suggested by Table I for the simulation results.

6. STARR-DCS ADAPTATION FOR SPATIALLY INHOMOGENEOUS TRAFFIC

We have presented a framework for scenarios where consumption and production are roughly homogeneously distributed within the network. In addition, not only STARR-DCS, but all previous works proposing multi-replication DCS systems also assume that consumption and production intensities are spatially homogeneous. However, this premise cannot be accepted in many scenarios in which consumption/production queries/events are more concentrated in some parts of the network than others. Therefore, in this section we consider a different scenario where production and/or consumption intensities are spatially inhomogeneous.

STARR-DCS could easily be extended to this new scenario by using the measurement period in which all replicas exchange local traffic information. This information is used to spawn new replicas in those areas with more traffic and prune those replication nodes that are located in areas where there is little traffic.

In particular, in this section we consider the case in which consumption dominates production (discussion would be similar for the opposite case). In this case the production traffic is global since each production event is received by all the replication nodes. However, consumption traffic remains local between a consumer and its closest replica. Therefore, we should consider the localization of the consumption traffic to decide which areas of the network need more replicas, and which existing replicas could be pruned. When the replication nodes exchange traffic statistics they know what is the consumption traffic of the different replication nodes. Therefore, each node can compute how many replicas should be assigned to the region supported by the current replica depending on its measured consumption traffic.

For instance, let us assume a scenario with 5 replicas in the current epoch, which have exchanged their measured consumption and production traffic during the measurement period. Suppose the percentage of consumption traffic for each of the 5 replicas is: 0, 10, 40, 0 and 50 percent respectively. Under this traffic pattern we propose a new distribution for the 5 replicas in the current epoch: the first and fourth replicas are pruned since they do not see any consumption traffic, the second replica deserves to be converted into 0.5 replicas since it is only receiving a 10% of the overall consumption traffic, so we keep the second replica without pruning. The third replica should be allocated a 40% of the total of 5 replicas, which is 2 replicas, therefore a new replica will be spawned by generating locations with the hash function until the output coordinates are closer to the second replica than ro any of the other replication nodes. Finally the fifth replica deserves 2.5 out of the 5 replicas, so 2 new replicas are spawned whose location are closer to the fifth replica than to any of the initial replication nodes.

It must be noted that in the above example we have decided to round the numbers of replicas, which means that if a node is assigned a value ≥ 0.5 replicas we decide to assign 1 replica (in case 3.5 we assign 4 replicas), thus if the value is < 0.5 we assign 0 replicas. In many cases, the number of replicas after the heterogeneous adaptation could be different to the initial value (e.g. in the previous example after the adaptation there would be 6 replicas in the scenario instead of 5). A more sophisticated algorithm could be used in order to better fit the initial number of replicas.

Note that each of the initial replicas can locally compute the location of the new replicas after the measurement period without incurring any extra traffic with respect to the solution for the homogeneous case, except for the messages employed to notify the new spawned replicas its new role. This overhead is very low because the new replicas are notified by the ones that are spawning them that are close in the field.

We propose to perform this mechanism once per epoch, making the system adaptive to changing spatial distributions of consumption/production.

Consumers and producers need to be notified of the locations of the new replicas because now the closest replica could be one of the spawned ones or one of the pruned ones. In the original STARR-DCS framework producers and consumers were notified of the duration of the current epoch and the number of replicas after the measurement period. That notification could be used to also re-direct the associated consumers and producers to the spawned replicas location. Therefore, redirecting consumers and producers to the new spawned replicas needs not produce any overhead.

Next, we compare this new adaptive heuristic for scenarios with a spatially inhomogeneous traffic distribution with the original STARR-DCS proposal that assumes homogeneous traffic distribution. We will refer to the first one as *Adaptive STARR-DCS*.

6.1. Performance Evaluation

This section explores the benefits of spatial adaptation where consumption and/or production intensity is spatially inhomogeneous. To that end we compared the Adaptive STARR-DCS and the basic STARR-DCS in terms of overall network traffic. Thus, we compare both approaches under a same traffic generation pattern and get the overall network traffic in one simulation cycle. In addition, we also measure the network lifetime in long term simulations by showing the number of dead nodes over the time for both approaches.

6.1.1. Overall network traffic comparisons. In order to evaluate the overall network traffic, we use an ad-hoc simulator that allows us to deal with large-scale WSANs and achieve a good scalability degree.

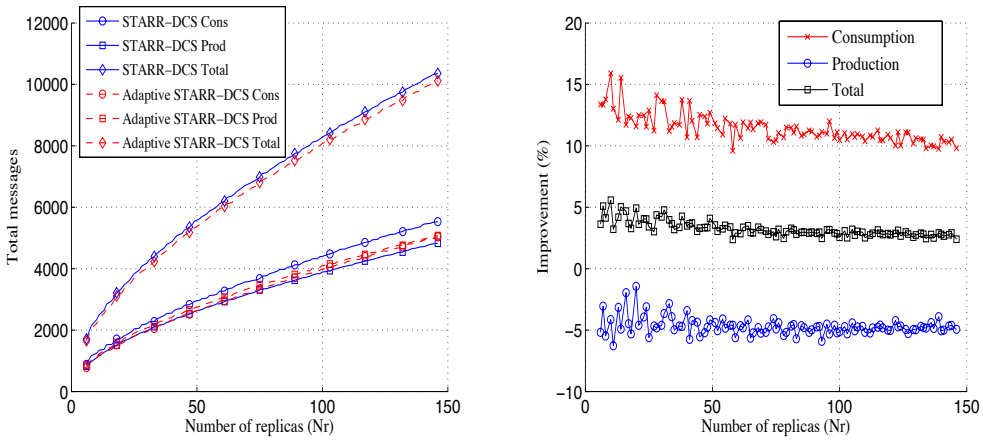
To create the heterogeneous traffic distribution we divide a square WSAN into 4 smaller squares and assign a portion of the total traffic to each of them. Then, we define two different set of values, $\delta = [\delta_1, \delta_2, \delta_3, \delta_4]$ that refers to the portion of consumption traffic associated to each region and $\gamma = [\gamma_1, \gamma_2, \gamma_3, \gamma_4]$, that refers to the intensity of production traffic assigned to each small square. We evaluate via simulation the savings of using the adaptive approach. It must be noted that in this subsection we just compare the results over one cycle, and as it was mentioned above we are focused on the case when consumption dominates production. We consider different degrees of inhomogeneity in our simulations:

— $\delta_a = [0.25, 0.25, 0.25, 0.25]$ (see Figure 15 and Table IV)
— $\delta_b = [0.2, 0.2, 0.4, 0.2]$ (see Table IV)
— $\delta_c = [0.15, 0.15, 0.55, 0.15]$ (see Table IV)
— $\delta_d = [0.1, 0.1, 0.7, 0.1]$ (see Figure 16 and Table IV)
— $\delta_e = [0.05, 0.05, 0.85, 0.05]$ (see Table IV)
— $\delta_f = [0, 0, 1, 0]$ (see Figure 17 and Table IV)

The first case actually refers to a homogeneous consumption traffic distribution since all the zones are assigned with the same portion of consumption traffic. Later we increase the heterogeneity degree until the case in which all the consumption traffic is concentrated in one of the four quadrants. In addition, we keep the production intensity homogeneous across the network with $\gamma = [0.25, 0.25, 0.25, 0.25]$.

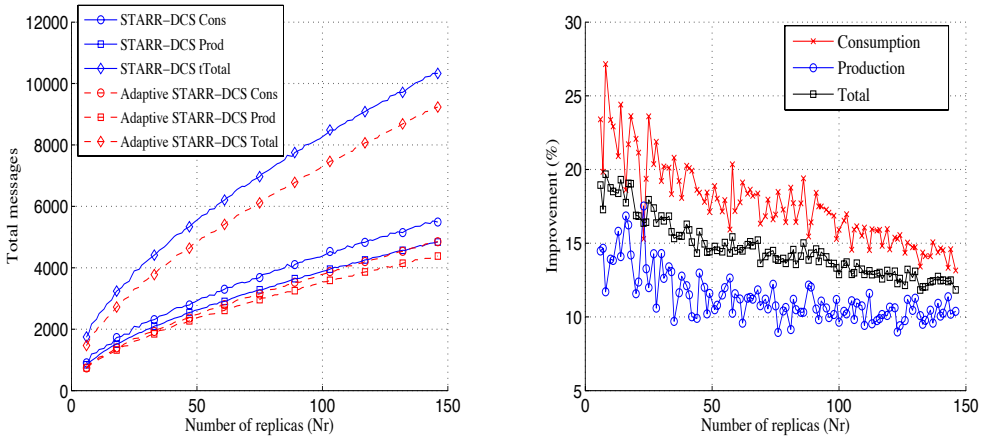
We simulated a scenario with: an area $A = 1000 \times 1000$ m^2, N=5000 nodes, a λ_c/λ_p ratio ranging from 4 to 100, measuring the overall traffic with the overall number of messages generated in the network. We have run 50 iterations for each ratio value and represented the average overall network traffic.

In order to run the simulation we consider periodic traffic generation by each consumer and each producer. That is, in one cycle (time unit) every consumer sends a query (and receives the reply) and every producer generates an event. Therefore, we get the different λ_c/λ_p ratios by keeping constant the number of producers and increasing the number of consumers.



(a) Traffic (b) Improvement

Fig. 15. Adaptive STARR-DCS vs basic STARR-DCS traffic comparison with $\delta = [0.25, 0.25, 0.25, 0.25]$ (A=1000x1000 m^2, N=5000 nodes, Tx=50 m.)

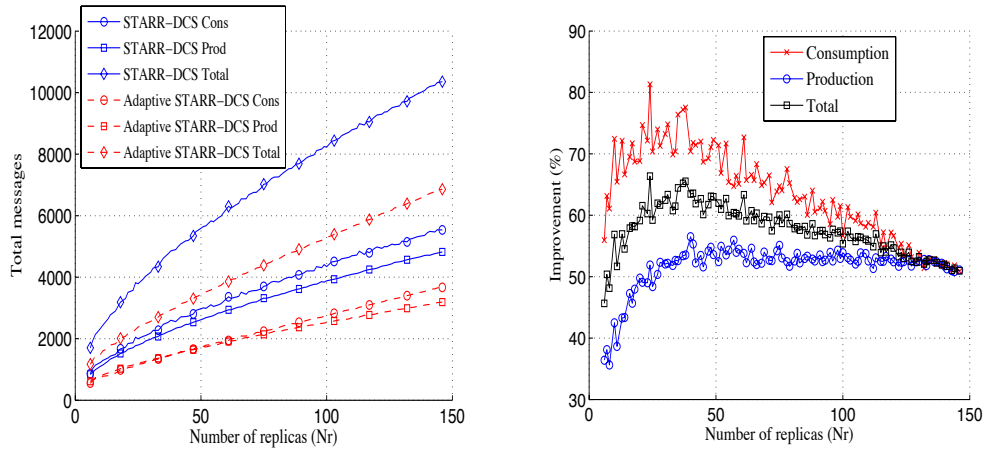


(a) Traffic (b) Improvement

Fig. 16. Adaptive STARR-DCS vs basic STARR-DCS traffic comparison with with $\delta = [0.1, 0.1, 0.7, 0.1]$ (A=1000x1000, N=5000, Tx=50).

All the graphs show in the x axis the optimal number of replicas (N_r) provided by the model for a particular ratio λ_c/λ_p, and in the y axis, the Figures 15(a), 16(a) and 17(a) represent the consumption, production and overall traffic for each mechanism measured in overall number of messages, and the Figures 15(b), 16(b) and 17(b) the relative improvement for consumption, production and overall traffic, when the adaptive approach is applied.

First of all, we highlight that even in the homogeneous traffic scenario (δ_a), by applying the novel heuristic it is possible to outperform the results of the basic STARR-DCS. The reason is that, even if we are in a homogeneous deployment, since replicas are located at random some of them may receive a higher consumption rate while others a lower or no traffic, e.g., replicas located at the borders of the network. In such cases, applying the



(a) Traffic (b) Improvement

Fig. 17. Adaptive STARR-DCS vs basic STARR-DCS traffic comparison with $\delta = [0, 0, 1, 0]$ (A=1000x1000, N=5000, Tx=50).

adaptive solution allows us to prune useless replicas and/or spawn new ones in those areas experiencing a higher consumption.

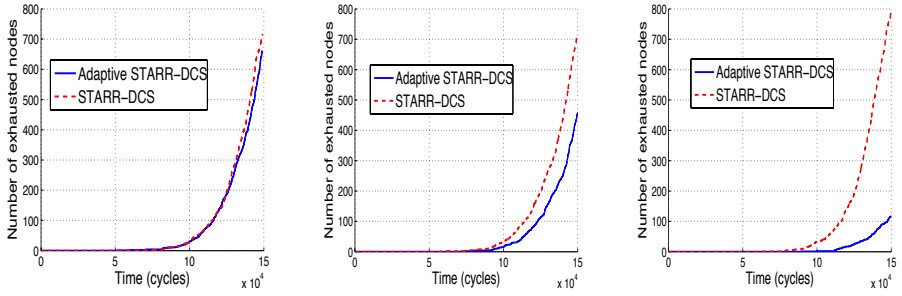
In addition, when the spatial consumption distribution becomes more and more heterogeneous the adaptive approach provides a more relevant improvement for the overall network traffic, reaching peaks above a 60% in the most extreme case (δ_f).

Another conclusion is that in medium/high inhomogeneity cases, the improvement is better for a low number of replicas and decreases when the number of replicas grows. This happens because DCS solutions are more sensitive in cases where there are fewer number of replicas since a bad disposition of a few replicas leads to very bad results, whereas if the few replicas are located taking into account the consumption load, the overall traffic is reduced a lot.

Table IV. Query consumption and event production path lengths comparison in number of hops for Adaptive STARR-DCS and basic STARR-DCS

δ	**Consumption Queries Avg. Path Length basic — adaptive**	**Production Events Avg. Path Length basic — adaptive**	**Consumption Difference**	**Production Difference**
δ_a	5.63 — 5.11	27.08 — 29.23	9.81%	-6.72%
δ_b	5.72 — 5.22	27.33 — 28.36	9.5 %	-3.63%
δ_c	5.95 — 5.23	27.77 — 25.94	13.74%	7.07%
δ_d	5.77 — 5.15	28.07 — 24.31	12.1 %	15.47%
δ_e	5.57 — 4.39	27.29 — 21.90	27.3 %	24.63%
δ_f	6.79 — 4.85	28.6 — 24.77	40.06%	15.46%

Table IV shows the average number of hops for consumption queries and production events (that also includes the replication tree). The results in the table show the traffic reduction when applying the adaptive heuristic. Consumers' communication paths are much shorter when applying the adaptive STARR-DCS solution in inhomogeneous traffic conditions. In particular, the higher the spatial consumption heterogeneity, the larger the improvement. This occurs because we are placing the replication nodes closer to consumers. Although the adaptive solution does not pay attention to the producers to re-allocate the replication nodes, in medium/high heterogeneity scenarios the production paths are also shortened. However, in low heterogeneity scenarios production paths are slightly shorter in the original



(a) δ=0.25,0.25,0.25,0.25 (b) δ=0.1,0.1,0.7,0.1 (c) δ=0,0,1,0

Fig. 18. Number of dead nodes over the time when comparing the Adaptive STARR-DCS and basic STARR-DCS solutions under different heterogeneous traffic distributions (A=1000x1000 m^2, N=2500 nodes, Tx=30 m, Battery= 10^6 message, E_{th} = 2*10^5 messages).

STARR-DCS approach, but the overall traffic can still be reduced by the spatially adaptive heuristic.

6.1.2. Lifetime. Next we compare the adaptive approach to the original STARR-DCS framework in terms of lifetime when there is a inhomogeneous distribution for consumption in the network. We measure the number of dead nodes over time for both approaches subject to the same traffic pattern.

We first evaluate a static traffic scenario in which the traffic pattern remains constant across the simulation. In this case, the evolution of dead nodes is quite similar in both approaches, which is an obvious result. If most of the traffic is localized in a particular area of the network, the nodes in that area run out of battery quicker than the rest of the nodes, and it does not matter if we update the replica location to that area, it still is the area with most traffic and the nodes there are the ones likely to exhaust their batteries earlier.

Therefore, what is interesting is to compare both solutions when the traffic is dynamic, that is, the zone with heavier consumption and/or production intensities changes over the time. In this case, being adaptive to inhomogeneous traffic distribution has an impact on extending the nodes lifetime.

We have performed a long-term simulation under different heterogeneity conditions. In all the cases we have simulated a WSAN with a grid node deployment of area A=500 $\times$ 500 m^2, N=2500 nodes with a transmission range Tx=30 m. We deploy 300 consumers and 100 producers. The epoch message threshold that initiates an epoch transition is 200000 messages. The measurement period at the beginning of an epoch when traffic statistics are measured lasts 10 cycles. We generated consumption mobility by changing the consumer nodes every 5000 cycles. The simulation duration was 150000 cycles. Finally, we assume that a node runs out of battery after 1 million messages (sent+received), thus there are nodes that die and do not participate in the network at some point in time during the simulation.

For the consumption traffic we have used heterogeneity distributions defined above as δ_a, δ_d, and δ_f, but randomly selecting one of the quadrants to apply the heavier traffic portion every 5000 simulation cycles. A homogeneous distribution for the production traffic was employed.

Figure 18 shows the number of dead nodes over time for the adaptive and basic STARR-DCS approaches. The first conclusion is that the adaptive approach allows nodes to extend their lifetimes even when it is applied in a scenario with a fully homogeneous traffic pattern (see Figure 18(a)). In addition, the higher the heterogeneity, the fewer nodes die when using the adaptive solution. Moreover, the original STARR-DCS behaves worse under high spatially inhomogeneous traffic patterns.

Therefore, incorporating the sptailly adaptive heuristic to STARR-DCS does not only achieve better performance when there is a substantial traffic heterogeneity, but it also extends nodes' lifetimes under homogeneous conditions.

7. CONCLUSIONS

This paper demonstrates that static DCS systems are highly unfair in terms of energy consumption. Those nodes assigned with the role of home nodes and its surrounding neighbours will naturally spend much more energy than other nodes in the network. Therefore, in order to extend the network lifetime is mandatory to fairly distribute the energy consumption across the network nodes. Then, this paper first demonstrates that placing multiple replication nodes at random in a DCS system outperforms previous work in the literature based on a single rendezvous point, grid-structured or uniform replication node placement. Furthermore, we have shown that equalizing the energy burdens across the network, by means of changing the set of replication nodes over the time, achieves a huge improvement in the WSAN lifetime. These two conclusions have been validated in large WSANs by simulation, and a testbed with 20 motes. Finally, this work is the first to explore networks where events and queries generation are not homogeneous across the network. We have proposed a heuristic that effectively reduces the traffic in the network and leads to use shorter communication paths.

REFERENCES

Ahn, J. and Krishnamachari, B. 2006. Fundamental scaling laws for energy-efficient storage and querying in wireless sensor networks. In *MobiHoc '06*. ACM, New York, NY, USA, 334–343.

Ahn, J. and Krishnamachari, B. 2009. Scaling laws for data-centric storage and querying in wireless sensor networks. *IEEE/ACM Trans. Netw. 17,* 4, 1242–1255.

Baccelli, F. and Bordenave, C. 2007. The radial spanning tree of a poisson point process. *Annals of Applied Probability 17*, 305.

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

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

Baek, S. J. and de Veciana, G. 2007. Spatial model for energy burden balancing and data fusion in sensor networks detecting bursty events. *IEEE Trans. on Information Theory 53,* 10, 3615–29.

Baek, S. J., de Veciana, G., and Su, X. 2004. Minimizing energy consumption in large-scale sensor networks through distributed data compression and hierarchical aggregation. *IEEE Journal on Selected Areas of Communications 22,* 6, 1130–1140.

Cuevas, A., Urueña, M., Romeral, R., and Larrabeiti, D. 2010. Data centric storage technologies: Analysis and enhancement. *Sensors 10,* 4, 3023–3056.

Dietrich, I. and Dressler, F. 2009. On the lifetime of wireless sensor networks. *ACM Trans. Sen. Netw. 5,* 1, 1–39.

Ee, C. T., Ratnasamy, S., and Shenker, S. 2006. Practical data-centric storage. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*. USENIX Association, Berkeley, CA, USA.

Jain, R. K., Chiu, D.-M. W., and Hawe, W. R. 1984. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Tech. rep., Digital Equipment Corporation. September.

Joung, Y.-J. and Huang, S.-H. 2008. 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*. Springer-Verlag, Berlin, Heidelberg, 237–251.

Karp, B. and Kung, H. T. 2000. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00*. ACM, New York, NY, USA, 243–254.

Liao, W.-H., Shih, K.-P., and Wu, W.-C. 2010. A grid-based dynamic load balancing approach for data-centric storage in wireless sensor networks. *Comput. Electr. Eng. 36,* 1, 19–30.

Moller, J. 1994. *Lectures on random Voronoi tesselations*. Springer-Verlag.

Ratnasamy, S., Karp, B., Shenker, S., Estrin, D., Govindan, R., Yin, L., and Yu, F. 2003. Data-centric storage in sensornets with ght, a geographic hash table. *Mob. Netw. Appl. 8,* 4, 427–442.

Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., and Shenker, S. 2002. 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*. ACM, New York, NY, USA, 78–87.

Shenker, S., Ratnasamy, S., Karp, B., Govindan, R., and Estrin, D. 2003. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev. 33,* 1, 137–142.

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

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