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

ÁNGEL CUEVAS and MANUEL URUEÑA, Universidad Carlos III de Madrid
GUSTAVO DE VECIANA, University of Texas, Austin
ADITYA YADAV, University of Massachusetts Amherst

Abstract: This article presents a novel framework for data-centric storage (DCS) in a wireless sensor and actor network (WSAN) that employs a randomly selected set of data replication nodes, which also change over 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 extend a WSAN's lifetime by at least 60%, and up to a factor of 10× depending on the lifetime criterion being considered. The feasibility of the proposed framework has been validated in a prototype with 20 resource-constrained motes, and the results obtained via simulation for large WSANs have been also corroborated in that prototype.

Categories and Subject Descriptors: C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*Wireless communication; Distributed networks*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Wireless sensor and actor network (WSAN), data-centric storage (DCS), random replication, epoch, optimization

ACM Reference Format:

Cuevas, Á., Urueña, M., de Veciana, G., and Yadav, A. 2013. STARR-DCS: Spatio-temporal adaptation of random replication for data-centric storage. ACM Trans. Sensor Netw. 10, 1, Article 14 (November 2013), 37 pages.

1. INTRODUCTION

In this article, we consider a simple framework to build a distributed information delivery service for one or more applications running over a wireless sensor and actor network (WSAN). Each application is modeled as a (randomly) distributed set of *producer* and *consumer* nodes, for example, sensors or actuators that exchange information by relaying packets across neighboring nodes. We assume that producer and consumer nodes do not have explicit knowledge of each other, but are just aware of the name(s) of the application(s) in which they are participating. This makes it 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, 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 the 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 article, 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 $hash(APP \oplus epoch \oplus i) \ \ \forall i \in [1, N_r]$, 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 that 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 trade-off 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, that is, it is critically dependent on the traffic associated with the application.

In the case where the consumption intensity dominates production one, data is copied across all replication nodes, whereas in the opposite case producers store data solely at

the closest rendezvous node, and so consumers query all 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 experiences a higher traffic load associated with supporting consumption and production, and thus its energy reserves are 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 proposed framework is thus highly flexible, yet also presents challenges in terms of optimizing adaptation to application's traffic.

Finally, we have implemented the proposed framework on a set of 20 motes for the case where consumption dominates production. Our implementation validates in a real small-scale deployment most of the expected outcomes from our theoretical work.

1.1. Related Work

Data-centric storage is inspired by distributed hash table (DHT) mechanisms, like the well-known Chord [Stoica et al. 2001] or KAD [Maymounkov and Mazières 2002]. DHT solutions are based on a limited ID space shared by nodes and resources. A node finds its DHT ID by applying a hash function over a *key* (e.g., the node IP address). Similarly, a resource is assigned an ID following the same operation, but in this case, the key is a property of the resource (e.g., the file name). The closest node (based on its node ID) to that resource ID is the responsible for storing either the resource itself or a pointer to the resource. Therefore, when some node wants to find a resource, it first looks for the resource's ID by sending a query through the DHT that reaches the node responsible for the resource, which finally replies back to the source node with the resource or a pointer to it.

DCS proposals operate similarly to DHTs. In DCS, nodes use a hash function over the application name or event type (i.e., a resource in DHT) to obtain some coordinates (i.e., the resource ID in DHT). The closest node to those coordinates is responsible for storing and serving the information related to that application (i.e., the responsible node for the resource in DHT).

In Cuevas et al. [2010], we presented a detailed survey discussing the main work on data-centric storage. Next we only discuss related works that are 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 article considers the use of a single replication node.

Approaches using multiple rendezvous (replication) nodes were subsequently proposed [Ratnasamy et al. 2002, 2003; Cuevas et al. 2010; Joung and Huang 2008; Ahn and Krishnamachari 2006], yet these studies place replicas in a structured manner, for example, 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, 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 any solution 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 the 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.

Ahn and Krishnamachari [2006] 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 5, using the number of replicas suggested by this paper leads to a much worse performance than ToW, QAR, and Random Replication.

Most of these works assume a square sensor field. If the sensor field is not square, for example, rectangular or some other irregular shape, these approaches [Ratnasamy et al. 2002, 2003; Cuevas et al. 2010; Joung and Huang 2008] could become much less efficient. By contrast, the approach proposed in this article 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 [Thang et al. 2006; Liao et al. 2010; Ahn and Krishnamachari 2009]. However, these works just focus on balancing the storage load and do not take into account energy considerations. In addition, they do not analyze what are the cost and implications of such changes and how it affects the network performance, as is done in this article.

1.2. Contributions

To the best of our knowledge, this article makes several novel contributions to the study of data-centric storage for wireless sensor and actor networks (WSANs).

—We propose STARR-DCS, a Spatio-Temporal Adaptation of Random Replication framework for Data-Centric Storage, which employs sets of randomly located replicas that can change over the time. The research contributions of the proposed solution rely on three main axes: (i) a mathematical analysis that establishes the theoretical basis to optimize the use of STARR-DCS; (ii) a comprehensive evaluation of STARR-DCS in large WSANs using a simulation environment that allows us to compare STARR-DCS with previous proposals in the literature; (iii) an implementation of STARR-DCS in resource-constrained commercial motes that demonstrates the feasibility of the different algorithms and protocols designed for our framework.
—Random Replication is a novel mechanism that places replication nodes randomly in the network. It provides three advantages as compared to previous proposals that

use a deterministic placement of replication nodes (e.g., grid). It is simpler from a computational point of view, it can be used independently of the network shape, and, more importantly, it enables using STARR-DCS in networks without geographic information. To the best of our knowledge, all previous multi-replication DCS proposals were designed to work in scenarios that require geographic information. Finally, our performance evaluation demonstrates that Random Replication is the most efficient DCS replication mechanism in terms of traffic overhead.

—We propose and validate a simple model to determine the optimum number of randomly-placed replicas in order to minimize the overall network traffic and its associated energy consumption, given the measured intensities for production and consumption of an application.

—Starting from the previous model, we perform a mathematical analysis that assesses the utilization of other resources, such as storage, which allows us to evaluate whether memory requirements are sufficient when multiple applications share network resources, or if the amount of replication should be constrained due to a limited memory capacity in nodes.

—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. We achieve this by changing replicas over the time, which introduces a number of challenging issues that have been solved by means of new protocols and algorithms.

—We propose various protocols and algorithms to implement STARR-DCS: (i) We divide the time into epochs that allow consumer and producer nodes to compute the replication nodes at any particular time. (ii) We define epoch transitions based on a threshold for the maximum traffic sent+received by a replication node while playing that role. This further equalizes the energy consumed by nodes acting as replicas independently of whether they play such role under a peak traffic period or not. (iii) We propose a mechanism that allows all nodes that participate in an application to smoothly move from an epoch to the next one. For that purpose, we rely in relative time synchronization (avoiding global synchronization that is very complex and costly for a WSAN) to establish epoch's deadline. (iv) We propose using of a *Meta-Information Service* that supports all applications running concurrently in the WSAN. This service enables efficient bootstrapping of new nodes and new applications, while addressing key fault-tolerance requisites for such networks.

—We simulate STARR-DCS in large WSANs and demonstrate that (i) Random Replication is the most efficient mechanism in terms of network traffic, and (ii) STARR-DCS in a large WSAN extends the lifetime at least by 60% as compared to previous proposals in the literature that rely in static replication points. This enhancement can go up to 10× under certain conditions.

—We have implemented STARR-DCS on resource-constrained commercial motes to validate the practical feasibility of the proposed framework. In addition, we have performed several experiments in a 20-mote prototype that validate the main analytical model and simulation results presented here, and show that they are also applicable in a small-scale network.

1.3. Paper Organization

The remainder of this article is structured as follows: Section 2 presents STARR-DCS and describes its operation in detail. Section 3 describes the implementation details of STARR-DCS in commercial motes. The analytical model employed to analyze and optimize resource utilization is described in Section 4. Section 5 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

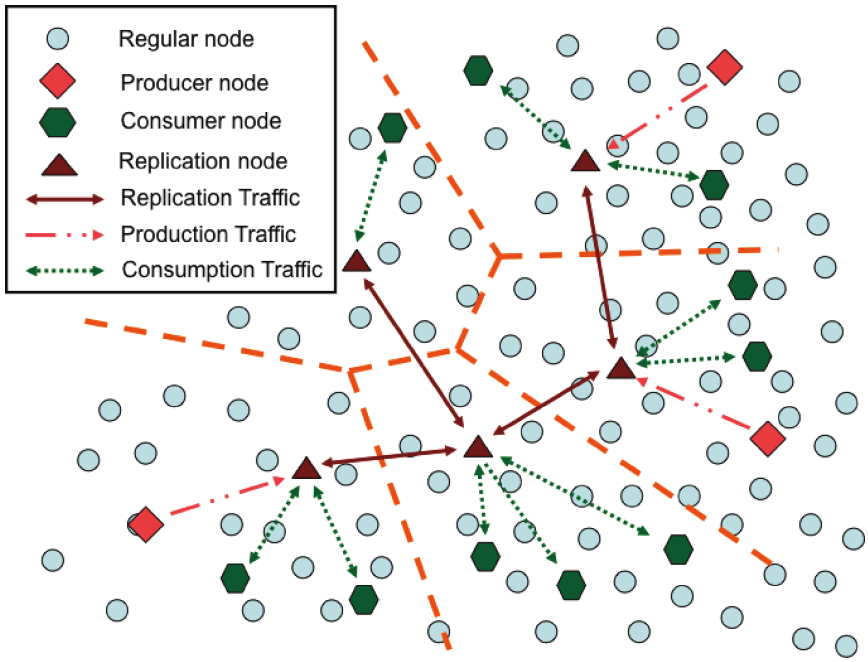


Fig. 1. Example of data-centric storage WSAN with five randomly-placed replicas.

the evaluation of STARR-DCS in a real testbed in Section 6. Finally, Section 7 offers concluding remarks and discusses the promise of the proposed approach.

2. STARR-DCS OPERATION

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

In the following text, we introduce the functionality required in our proposal for the case where consumption traffic dominates production one, also illustrated in 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 4). To simplify the description, we start assuming that this information is known by every node participating in a given application.

Random Selection of Rendezvous Point Locations. Any node in the network that knows *APP*, e, and N_r is able to compute the rendezvous points' locations at any particular time. For that, a node just needs to compute the following hash operation: $hash(APP \oplus e \oplus i)$, $\forall i \in [1, N_r]$ that generates N_r random locations within the network.

A hash function produces as its output a random-like bitstring (e.g., 128 bits) that can be used to generate one or more coordinates in the physical space covered by the WSAN. For instance, if we think of a square bidimensional surface, we can employ the first half of the bitstring to compute the X coordinate (by using a modulo operation to map the bits' value to the actual network dimension), and the second half to compute the Y coordinate. A similar procedure could be employed to compute a radius length and an angle in a circular surface. Next, the closest node to each one of the locations generated using the hash function becomes a rendezvous node. Note that there are several ways of finding the closest node to a given location like the one proposed in GHT [Ratnasamy et al. 2003], but for simplicity, in this section, we will equivocate the rendezvous nodes with the associated hashed locations.

Producers and Consumers Functionality. Suppose a producer (consumer) node generates an event (query) related to *APP*. Such a node 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: $hash(APP \oplus e \oplus i), \quad \forall i \in [1, N_r]$. Once a producer/consumer node determines the closest rendezvous point, it forwards a message/query to that location, that is, to the closest replication node r_i to its location. In the consumption case, the rendezvous node just responds with the suitable data to the corresponding query. This replication location will be used for some time, so producers and consumers may cache the replication points' coordinates, avoiding its recomputation for every query/event associated with APP.

Creating a Tree to Replicate Data over Rendezvous Nodes. In 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 replication nodes (if any) to which it should forward new data. Since all rendezvous nodes know all 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 only contains the root node).
—$\mathcal{R}$. The set of rendezvous nodes to be reached, which initially contains all rendezvous nodes except the root node: $\mathcal{R} = \{r_2, \ldots, r_{N_r}\}$.
—$\mathcal{F}$. The set of rendezvous nodes to which the current rendezvous node running the algorithm should forward the event, which is initially empty: $\mathcal{F} = \emptyset$.

The algorithm proceeds as follows: the root node, r_1, computes which rendezvous node in $\mathcal{R}$ is the closest one to itself. Suppose it is r_2, then r_2 is removed from $\mathcal{R}$ and included in both $\mathcal{C}$ and $\mathcal{F}$, that is, $\mathcal{C} = \{r_1, r_2\}$, $\mathcal{R} = \{r_3, \ldots, r_{N_r}\}$, 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 shortest 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 shortest 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 node can similarly compute their associated forwarding sets $\mathcal{F}$. Note that if the preceding 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 the 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.

Algorithm 1 exhibits the pseudocode to compute the forwarding nodes of a replication node r_i, assuming a scenario with N_r replication nodes.

ALGORITHM 1: Replication tree construction algorithm run by replication node r_i to know which 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_Nr}.
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
```

Changing the Set of Rendezvous Nodes. We define an *epoch* as the time between two consecutive changes in the set of replication nodes. In addition, we consider two events that could trigger epoch changes: (i) when a node serving as a replication node exceeds a certain 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, rendezvous nodes gather 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 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 must be retransmitted if necessary. 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 other replicas (messages containing local traffic measurements) and that its information has been received by all remaining replicas (acknowledgement). Otherwise, errors in the estimation of the number of replication nodes to be used in the next epoch could lead to application inconsistencies.

This mechanism for triggering epoch transitions, based on a threshold for the total number of messages, can adapt to changing traffic characteristics. Thus, an application could suffer peak traffic periods in which the selected replication set would use short epochs, since it would quickly reach the established message threshold, and for those low-traffic periods where the epoch duration would be much larger, since the replication nodes would take longer to reach the message threshold. Since 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 needs only to determine which is the closest node in the subsequent set of replicas (associated locations). Then, such nodes can transfer, in parallel, their stored data to the new locations. Such messages would notify the recipients their (new) role as replicas for the application during 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 about when this change will be executed and the number of replicas to be used in the next epoch. 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 current replication nodes have estimated both values, they send a specific message or piggyback this information back to producers and consumers, respectively. Consumers and producers receiving the information can then cancel the flag until the beginning 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, we are able to enable 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:

—providing a bootstrapping mechanism for finding the current set of replicas for a given application;
—providing fault tolerance;
—providing an initialization mechanism to bring new applications online.

In order to solve the bootstrapping problem when a new node wants to participate as a producer or consumer in an application, we propose employing a meta-information service where each network application 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 current replicas about the time at which the current epoch will expire and the number of replication nodes to be used in the next

epoch, by using the flag mechanism just detailed. This meta-information service is just another application that may use the proposed replication framework itself.

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 neighbors what is the current Meta-Information epoch. Another aspect that should be taken into account is determining how the meta-information service knows that a given application is changing its epoch. The first replication node (i.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 because the information is vital to enable new nodes to participate in applications. Therefore, if the replication node selected to notify the epoch transition does not receive an 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 desynchronization. If a node fails accessing its closest replication node for a predefined number of times, it first 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 still contact the meta-information service, which 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 initialized on a STARR-DCS WSAN. When any 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 new application and assigns a random epoch number and a single replication node to that service. 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, which could be eventually performed by any network node.

Application Role in STARR-DCS. STARR-DCS has been designed to be used by applications of very different nature. Some generic examples are applications where consumers just need to retrieve the most recent event (e.g., real-time applications), applications where consumers operate fetching the last N events (e.g., one event from each producer node in the network), or applications where consumers need to retrieve all stored events (e.g., applications using historical information). The nature of a particular application will also determine the transition cost between two consecutive epochs. For instance, if consumer nodes only need to access the last N events, replication nodes just need to store that number of events, thus the epoch transition cost is determined by that N value.

In order to properly use STARR-DCS, each application is responsible for adequately configuring certain parameters of the different players, such as producers' event rate (if applicable), consumers' query rate, number of events retrieved from a query, and number of events that need to be stored by a replication node. Based on the generated

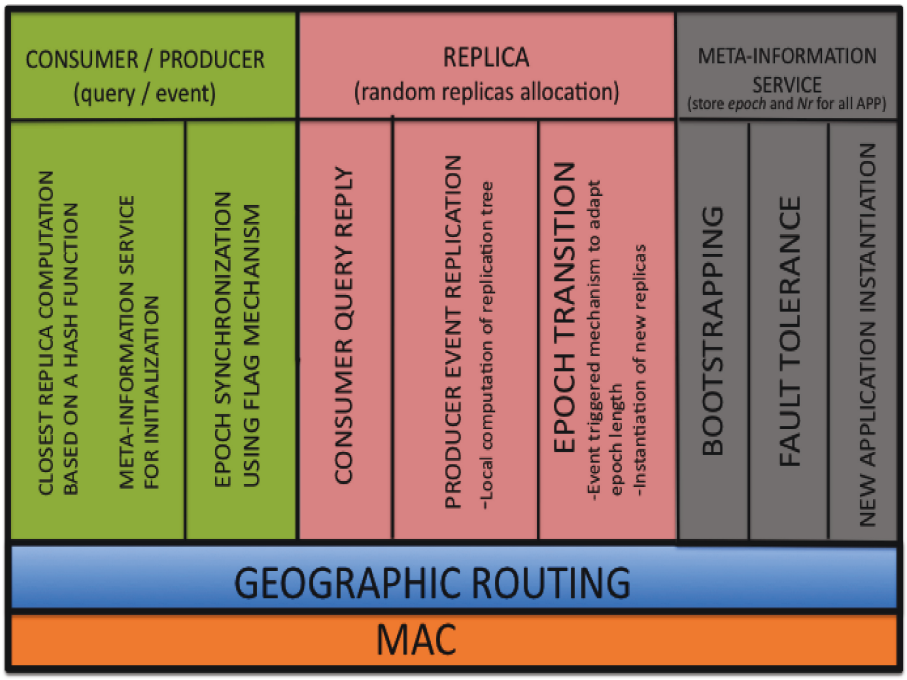


Fig. 2. Different roles that can be performed by a STARR-DCS node.

production and consumption traffic, STARR-DCS will establish the optimal number of replicas to be used and will balance the cost of being replica among all network nodes in order to minimize the overall traffic and maximize the network lifetime.

Note that a wrong configuration of these parameters could lead to worsen the network performance due to an unnecessary traffic overhead. For instance, if we know that in a particular application producer nodes roughly generate one event per hour, it does not make sense to configure consumers to generate one query per second, because this increases the traffic load but consumers will gather new data once per hour.

Testing different application types in STARR-DCS is out of the scope of this article and we will just select a generic case for our evaluation.

2.1. STARR-DCS without Geographic Information

Previous multi-replication DCS proposals [Ratnasamy et al. 2002, 2003; Cuevas et al. 2010; Joung and Huang 2008] need geographic information, since they rely on a deterministic rendezvous nodes placement that divides the network into regular sections (i.e., a grid structure). In contrast, an important novelty of the proposed STARR-DCS framework is that it is also suitable for networks that do not use geographic coordinates. Our algorithm basically selects random nodes to act as replicas, and this random selection could be based on a physical location, but it could also rely on random generation of node IDs. Then, it is straightforward to modify the hash function to compute a particular ID instead of a geographic location in the network (i.e., $ID\_r_i = hash(APP \oplus epoch \oplus i)$). The node with the closest ID to the hash function output would be selected as replica.

Coordinates-free networks need to fulfill two requirements to be able to implement STARR-DCS.

—The ID obtained from the hash function must be unambiguously assigned to a single node in the network (i.e., the node with the closest ID). For instance, if the node IDs are sequentially assigned in the network (e.g., from node 1 to node 100 in a network with 100 nodes) it is straightforward to define a hash function providing an output in that interval, and thus the replication nodes can be unambiguously identified.
—Consumer and producer nodes should be able to find the closest replication node, that is, they need to know or be able to compute the distance to all replication nodes. In this case, the distance metric could be the number of hops. For instance, in a wireless mesh network that implements a distance vector or a link state routing protocol each node knows all other nodes IDs and the distance in hops to reach them.

The fact that STARR-DCS can operate in networks without geographic information opens the possibility of using it in other wireless networks (i.e., wireless mesh networks) different than WSANs.

In the rest of the article we will study STARR-DCS under the classic DCS approach that uses geographic information and geographic routing. This allows us to directly compare our solution with competing solutions previously proposed in the literature.

3. STARR-DCS IMPLEMENTATION ON COMMERCIAL MOTES

We have implemented most STARR-DCS features on real motes. Our implementation supports an arbitrary number of replicas that change over the time and also includes 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 nodes involved in a particular application: consumers, producers, and replication nodes.

We have implemented STARR-DCS on 20 Jennic motes of two different models: JN-5121 (5×) and JN-5139 (15×). 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 that only includes 64kB of ROM.

A node can initially be assigned with one of these three roles: producer, consumer, or relay (in this 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 can generate events via two different mechanisms: (i) at a predefined rate, for example, 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 pressing a button.

We implemented a simple greedy forwarding algorithm[1] 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 neighbors, then it just assumes that is the closest one to the destination coordinates. The mechanism used to choose the nodes acting as replicas is to select the closest node to a randomly selected spatial location.

[1]We aim to validate the feasibility of STARR-DCS, and greedy forwarding is a simple yet valid routing protocol to achieve our goal. We could have implemented a more complex geographic routing, but this is out of the scope of our article since we do not aim to test the performance of WSNs routing protocols but the performance of STARR-DCS.

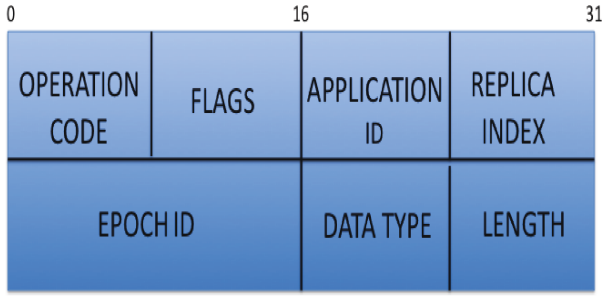


Fig. 3. STARR-DCS protocol header.

We have defined a common header to be used by all the protocol messages required to implement the STARR-DCS framework. Figure 3 shows all the different fields included in the header. Next, we describe each of these fields.

—*OPERATION CODE* defines the type of message (e.g., PUT, GET, GET REPLY, etc).
—*FLAGS* is used for special operations, that is, consumers and producers use one bit in this field to indicate that they do not yet know when the current epoch finishes and what is the number of replicas in the next epoch. Another bit is 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. In the case of a consumer query, it defines the number of events (i.e., data structures) to be retrieved.
—*REPLICA INDEX* identifies (if required) the replication node that is source of the message (i.e., that information is needed in all replication nodes in order to generate the replication tree in a distributed fashion).
—*EPOCH ID* specifies the current epoch of the source node, which is used for synchronization purposes.

Following, 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 agreed on the epoch expiration time and the number of replicas in the next epoch, the first replica (i.e., i = 1 in the hash operation) sends a PUT message to the closest 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 nodes set up a timer that ends at the end of the current epoch, until that moment they still serve 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 operation that requires a confirmation.
—*EPOCH ADVERTISEMENT* is used to notify producers 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 in.

—*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 predefined 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 those 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 four different applications in parallel on our testbed.

4. PERFORMANCE ANALYSIS

In this section, we propose a simple stochastic geometric model for the network that permits optimization of the large-scale system's parameters, that is, intensity of replication nodes. The approach follows the seminal work of Baccelli et al. [1997], Baccelli and Zuyev [1996] and our own work in applying this methodology to ad-hoc wireless networks (e.g., Baek et al. [2004] and 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, that is, 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 spatiotemporal 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 spatialtemporal 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 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 $\mathcal{A}$.

4.1. 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, that is, in region $\mathcal{A}$. Recall that in an ad-hoc wireless network traffic load cannot 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 $\frac{1}{2\sqrt{\lambda_r}}$, that is,

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

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 of $\lambda_p A$ 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 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

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

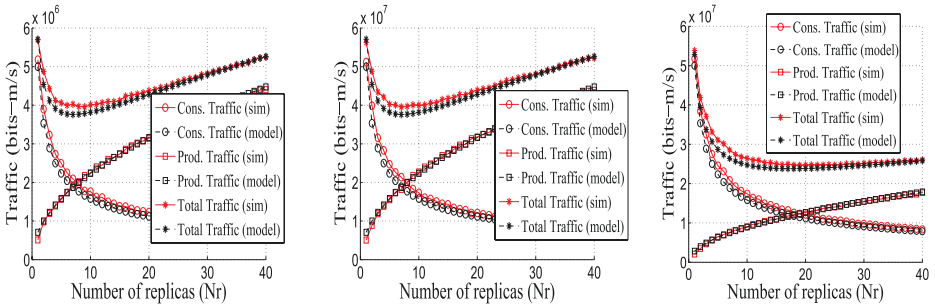
Remark 4.1. *Scaling Characteristics.* Roughly speaking, the optimal average number of replicas for the network covering an area A is given by

$$N_r^* = \left[\lambda_r^* A\right] = \left[\frac{\alpha \lambda_c}{\sqrt{2}\beta \lambda_p}\right] \text{ replicas,} \tag{1}$$

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 *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. Finally, we must notice that while Eq. (1) may provide a real number for the optimal number of replicas, in a real scenario, we will need to round it to select the actual number of replicas that will be deployed.

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 $N = 5{,}000$ nodes are randomly placed in a 1000×1000 m^2 region. We set $\beta = 100$ bits, assuming that producers periodically send the information to the closest replica without any acknowledgment. We set $\alpha = 200$ 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 4 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 $(500 \cdot 10^{-6}, 40 \cdot 10^{-6})$ $\frac{\text{events}}{\text{s·m}^2}$. 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 4(a) and 4(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 4(c)), there are several values around the optimal number



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

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 that follow straight lines. However, WSANs, which are the focus of this article, are multihop 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 5 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 it can be seen, when there is a low number of replicas, the model and the simulations are a good match. A small discrepancy occurs for high λ_c/λ_p ratios. However, as mentioned earlier, in the case this ratio is high, the overall cost is not very sensitive to the precise optimal value for the number of replicas.

Case 2(a): *Production-Dominates-Consumption* ($\lambda_c < \lambda_p$) *with Data Aggregation.* If the intensity of consumption is low relative to that of production, it may be preferable not to copy data across all replication nodes. Instead producers can store data solely at the closest replication node. Subsequently consumers should contact all replication points to gather the information. This could be done in several ways.

First, a symmetric model to that presented in the case of consumption-dominates-production could be also proposed. However, that model would assume that both queries and replies are sent through the replication tree once per branch, as it is done by ToW [Joung and Huang 2008] and QAR [Cuevas et al. 2010]. This can only be achieved if replies are aggregated, and such aggregation has implications that are beyond the scope of this article. In case that such aggregation happens, we present a symmetric model to the one in the consumption-dominates-production case.

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

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

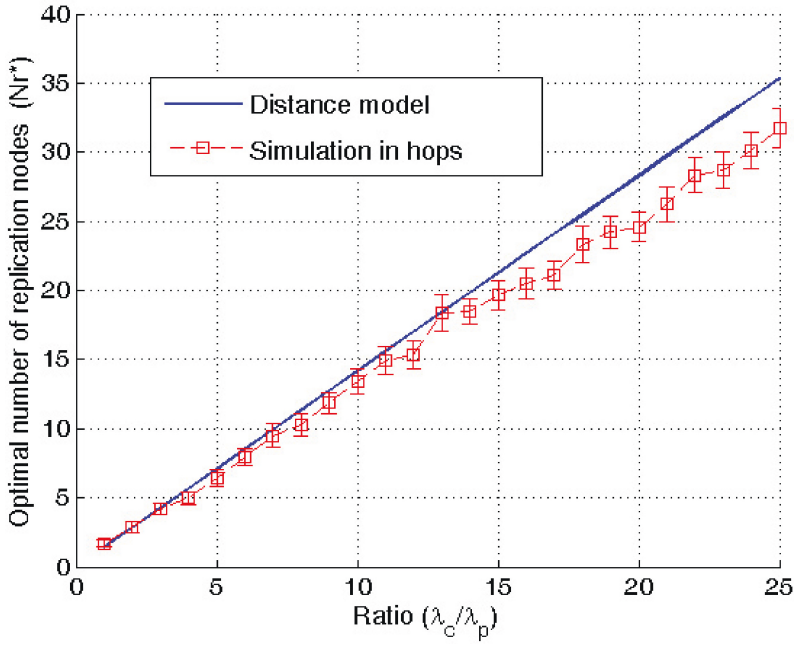


Fig. 5. Optimal number of replicas that minimizes the overall number of messages (A = 1000 × 1000 m^2, N = 5,000 nodes, Tx = 50 m).

The overall network traffic is modeled as

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

This can again be optimized to obtain the optimal spatial intensity for replicas λ_r^* given by

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

and the associated minimum overall network traffic is similar in form to Case 1.

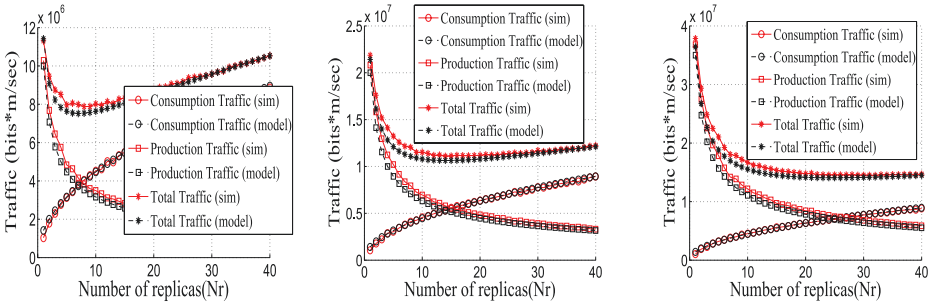
$$T\left(\lambda_r^*\right) = 2^{1/4}\sqrt{A}\sqrt{(\alpha\lambda_c A)(\beta\lambda_p A)} \text{ bits-m/s.}$$

Case 2(b): *Production-Dominates-Consumption* ($\lambda_c < \lambda_p$) *without Data Aggregation.* Many times aggregation could be a really complex task, since it requires additional state and processing inside the network. In addition, many applications cannot apply aggregation because they need all the produced data. For all those cases, we consider an alternative model where consumers contact all the replication nodes directly.

In this case the overall production traffic is

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

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 ($\lambda_c A$), and replicas ($\lambda_r A$). Thus the overall consumption traffic is



(a) N_c=10; N_p=200; N_r^*=7.07. (b) N_c=10; N_p=400; N_r^*=14.14. (c) N_c=10; N_p=700; N_r^*=24.64.

Fig. 6. Consumption, production, and total traffic generated by using different number of replication nodes for the case when production-dominates-consumption and the query replies are aggregated in the replication tree (A = 1000 × 1000 m^2, N = 5,000 nodes, α = 200 bits/query, β = 100 bits/event).

given by

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

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

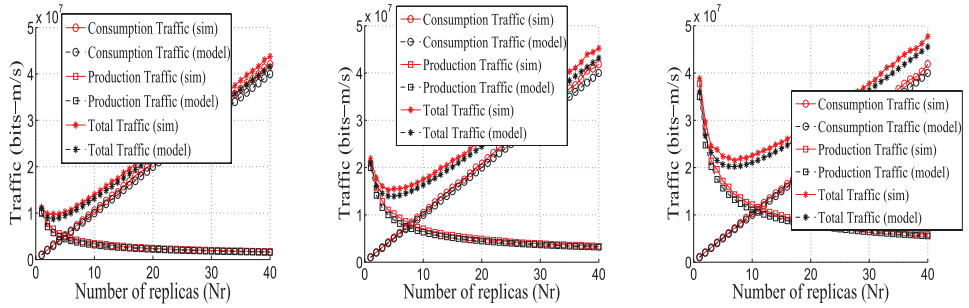
$$T\left(\lambda_r^*\right) = (\beta\lambda_p)^{2/3}(2\alpha\lambda_c)^{1/3}\frac{3A\sqrt{A}}{4} \text{ bits-m/s.}$$

Note that in this regime, the optimal intensity for replicas is a more complex function, that is, cubic of the ratio of production to consumption intensities, yet, in principle, still easily computable by sensors in real time.

Both models have been validated via simulation. We have used a scenario of area $A = 1000 \times 1000\ m^2$, where N = 5,000 nodes were randomly deployed. We used a factor α = 200 bits/query and β = 100 bits/event.

Figure 6 exhibits the overall consumption, production, and total traffic measured in bits-m/s obtained by the model and by simulation for the case when aggregation can be used. Then, three different (N_c, N_p) pairs have been evaluated: (10, 200), (10, 400), and (10, 700). They generate the next (λ_c, λ_p) pairs: ($10 * 10^{-6}$, $200 * 10^{-6}$), ($10 * 10^{-6}$, $400 * 10^{-6}$), and ($10 * 10^{-6}$, $700 * 10^{-6}$) $\frac{events}{s*m^2}$. The number of replicas employed varies from 1 to 40. Thus, the optimal average number of replicas for these cases is 7.07, 14.14, and 24.64, respectively. The model is very accurate to the results obtained via simulation as it was expected since this model is a symmetric one to the consumption-dominates-production case.

In addition, we used the same (λ_c, λ_p) pairs to evaluate the production-dominates-consumption model when consumers use unicast routing to access replication nodes directly. Figure 7 shows that, again, the proposed model is very accurate. As it was expected, when the replication tree cannot be used, the traffic grows since the routing



(a) N_c=10; N_p=200; N_r^*=2.92 (b) N_c=10; N_p=40; N_r^*=4.64 (c) N_c=10; N_p=700; N_r^*=6.74

Fig. 7. Consumption, production, and overall traffic generated by using different number of replication nodes for the case when production-dominates-consumption and consumers directly query all replication nodes without using a replication tree (A = 1000 × 1000 m^2, N = 5,000 nodes, α = 200 bits, β = 100 bits).

without aggregation is less efficient. Therefore, placing new replicas is more expensive, that is why the optimal number of replicas for the three (λ_c, λ_p) pairs are now 2.92, 4.64, and 6.74 respectively. These numbers are much lower than the optimal number of replicas when the replication tree is employed in both directions.

We can conclude that both models are very accurate and both make sense in practice, because, depending on the application, the utilization of the replication/aggregation tree can be feasible or not. If query replies cannot be aggregated, using the replication tree to forward individual replies is highly inefficient, and the best option is using direct unicast routes.

4.2. 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, that is, 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 that 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 an 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 $\sqrt{\mathrm{Var}(V)} = E[V](0.52)$ [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, that is,

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

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

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

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 previous sum as a Gaussian random variable, that is, $\sum_{i=1}^{m} X_i \sim N(\mu, \sigma^2)$, where μ and σ^2 correspond to the mean and variance of the sum. In particular,

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

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

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

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

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

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 have 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 consider the case where $b/d = 3$, that is, a node can simultaneously support at most three applications. For each scenario, we have evaluated the maximum number of replicas each application could use while ensuring that a typical node's saturation probability is lower than $\delta = 0.1$ both via simulation and with our analytical model. Figure 8 shows that the storage model and the simulation results are very close, showing a

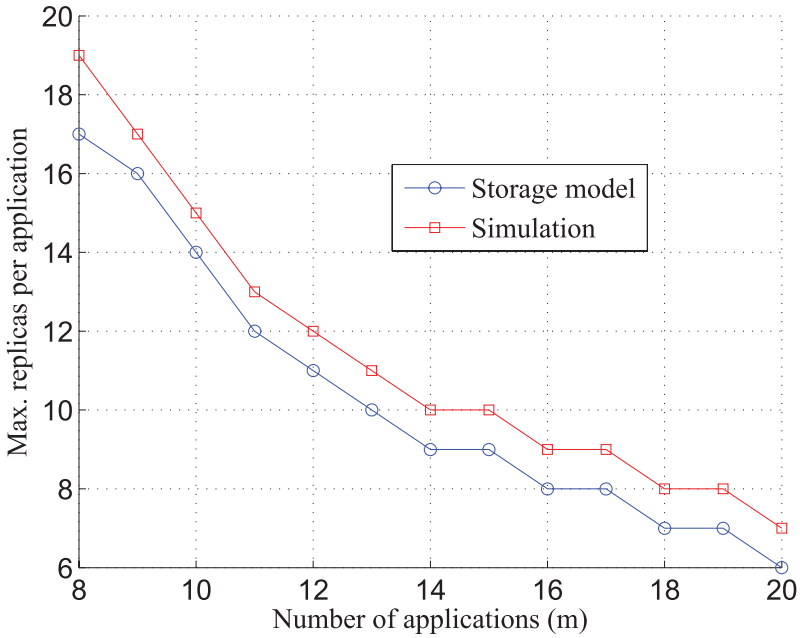


Fig. 8. Maximum number of replicas per application to keep the probability of node saturation below a 10% (A = 1000 × 1000, N = 100, b/d = 3, $\delta = 0.1$).

difference of just one replica in most 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.

4.3. 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.

4.4. Is Broadcasting Preferable?

Note that if the intensity of consumers is very high, producers could be tempted to simply broadcast new data to all nodes in the network. The overall traffic, T_b associated with broadcasting to all nodes in the network, $N = \lambda_n A$, can be modeled based on the length of the radial spanning tree reaching all nodes (some of which would be consumers).

$$T_b = \beta A^2 \lambda_p \sqrt{\frac{\lambda_n}{2}} \text{ bits-m/sec.}$$

Under this simple model, broadcasting would be favorable only if

$$T\left(\lambda_r^*\right) > \beta A^2 \lambda_p \sqrt{\frac{\lambda_n}{2}},$$

which is equivalent $\lambda_n < 4\lambda_r^*$. Thus, unless the optimal number of replicas is very high (i.e., on the order of 1/4 of the total number of nodes in the network), brute force broadcasting is not likely to be efficient. Note that this does not account for the so called wireless "broadcast advantage" whereby a node can send data to multiple nodes in a single transmission, and perhaps more efficient methods of realizing and modeling broadcasting. Still, the key here is that the optimal number of replication points would have to be very high indeed if broadcasting were to become more efficient than STARR-DCS.

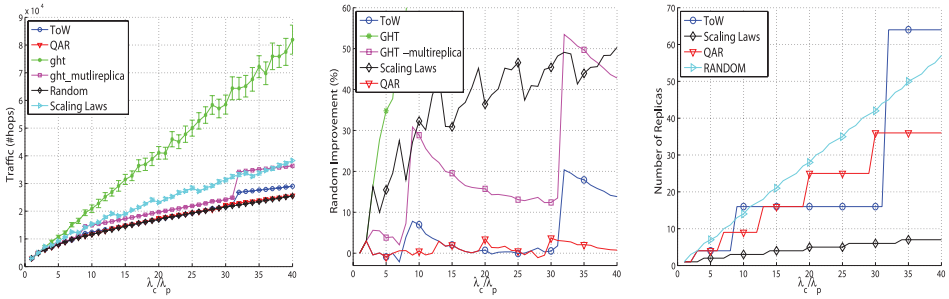
5. STARR-DCS EVALUATION: A SIMULATION STUDY IN LARGE WSANS

In this section, we consider two questions: (1) how selecting rendezvous nodes' locations at random compares to previous grid-based and uniform-based proposals; and (2) whether it is worthwhile to change the set of rendezvous nodes over time considering the associated overheads. We have developed a custom simulator that provides more scalability than standard ones, since it does not simulate wireless communications (i.e., PHY and MAC) other than transmission range. The use of this simulator allows us to easily test STARR-DCS in large WSANs scenarios containing thousands of nodes.

5.1. Random vs. Grid-Based and Uniform Replica Allocation

5.1.1. Evaluation of Network Traffic Costs. We have compared Random Replication that is the replication mechanism used in the proposed STARR-DCS framework with those proposals in the literature that are similar for consumption-dominates-production and production-dominates-consumption cases.

Consumption-Dominates-Production. For this traffic pattern ($\lambda_c > \lambda_p$), STARR-DCS can be compared to 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, 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.



(a) Overall traffic cost (#hops).

(b) Overhead traffic of other solutions when compared to Random.

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

Fig. 9. Random vs. ToW, QAR, Scaling-Laws, GHT, and GHT with multiple replicas for $\lambda_c > \lambda_p$ (A = $1000 \times 1000\ m^2$, N = 5,000 nodes, Tx = 50 m, $\alpha = 200$ bits, $\beta = 100$ bits).

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, $N = 5{,}000$ 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 and different λ_c/λ_p ratios. In order to get meaningful results, we use the number of hops traversed by all messages as the measure of overall traffic cost.

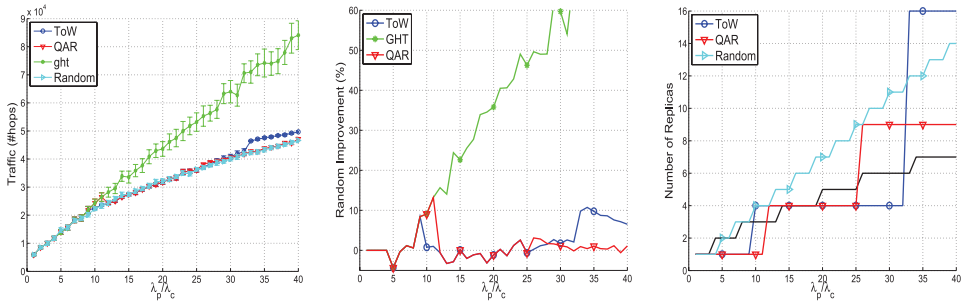
Figure 9(a) shows the overall network cost for all the analyzed approaches. Although the figure clearly shows that Random Replication is more efficient than GHT, Scaling-Laws, and GHT with multiple replicas, it is not easy to discern what the difference is between Random, QAR, and ToW. Towards this end, Figure 9(b) shows the network traffic improvement achieved using Random Replication as compared to the competing approaches (i.e., the extra traffic generated by the other proposals), and Figure 9(c) depicts the number of replicas used by each approach for each particular λ_c/λ_p ratio.

Random Replication reduces 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 50% when compared to Scaling Laws and GHT with multiple replicas, 15% to ToW, and 7% to QAR.

Production-Dominates-Consumption. Only ToW [Joung and Huang 2008], QAR [Cuevas et al. 2010], and the original GHT proposal [Shenker et al. 2003] with a single replica address scenarios where production traffic dominates consumption one ($\lambda_p > \lambda_c$). In addition, ToW and QAR assume aggregation in the replication tree. Therefore, in order to establish a fair comparison, we also assume aggregation in our solution. We use the same simulation parameters as in the previous case, but now the ratio λ_p/λ_c is the one ranging between 1 and 40.

Figure 10(a) shows the overall network cost for all the analyzed approaches. Figure 10(b) illustrates the network traffic improvement achieved by using Random Replication instead of the other approaches (i.e., the extra traffic created by the competing solutions), and Figure 10(c) depicts the number of replicas used by each proposal for each particular λ_p/λ_c ratio.

As it has happened in the previous case, Random Replication is the most efficient mechanism in terms of minimizing network traffic. In particular, Random Replication reduces the overall traffic by an average of 37%, 2%, and 1% as compared to GHT, ToW,



(a) Overall Traffic Cost (#hops) (b) Overhead traffic of other solutions when compared to Random (c) Number of replicas used for the different proposals: ToW, QAR, Scaling-Laws and Random

Fig. 10. Random vs. ToW, QAR, and GHT for $\lambda_p > \lambda_c$ (A = 1000 × 1000 m^2, N = 5,000 nodes, Tx = 50 m, α = 200 bits, β = 100 bits).

and QAR, respectively. In addition, we can find peaks that report a traffic reduction above 10% when Random Replication replaces ToW and QAR.

The main reason why our solution achieves a better performance in both cases is that it allows a finer granularity because the number of replicas being used can adapt better to application traffic load. That is, with Random Replication, 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 9(c)). 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.

5.1.2. Further Benefits of Random Replication. Surprisingly, our results show that Random Replication is the approach best minimizing the overall network traffic. It improves all previous approaches in the literature that are based in deterministic placement strategies like grid or uniform replication. Moreover, there are further reasons that make Random Replication a better option for DCS: (i) generating random locations is easier than computing a grid division to later allocate the replicas as required by QAR and ToW. (ii) Random Replication is flexible to be used under different network shapes (e.g., circular, irregular, etc.), whereas ToW and QAR are only applicable to shapes that can be easily divided into regular grids (e.g., squares or rectangles). (iii) More importantly, Random Replication is suitable to be used in scenarios where geographic coordinates are not available, as described in Section 2.1.

In a nutshell, Random Replication is simpler, more flexible to different network shapes, adaptable to networks without geographic information, and more cost effective. Therefore, these benefits validate our option of proposing Random Replication as the replication algorithm used in STARR-DCS.

5.2. Changing Replicas over 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 [Shenker et al. 2003; Ratnasamy et al. 2002, 2003; Cuevas et al. 2010; 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 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 $A = 300 \times 300$ m^2. Each node has a transmission range $Tx = 30$ m. In this case, we evaluate the scenario 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 receives one million messages. Finally, the threshold that determines the end of an epoch, E_{th}, is set to 300,000 messages (30% of the battery).[2]

For these simulations, we have used geographical routing based on greedy forwarding [Karp and Kung 2000]. When greedy forwarding fails, for example, due to routing holes, we use the shortest path from the node where the greedy forwarding stopped to the destination node.[3]

Time is measured in cycles in order to scale the simulations and 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.

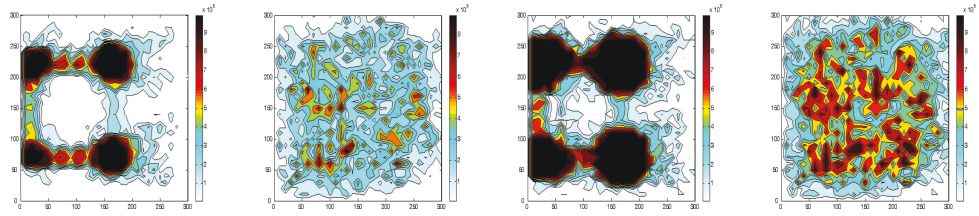
In order to measure the cost of an epoch change, we assume that the produced data has a mean lifetime of L cycles. Then, the average data stored at each replication node is $d = N_p L$ messages. L is set to 10 cycles for these simulations, thus the replication change is costly, because it means that ten messages per producer are moved from the old replicas to new ones. By having 100 producers, this means a total cost of $M = 1{,}000$ messages per epoch change.

Figure 11 shows the energy distribution map after a simulation time of 30,000 and 50,000 cycles. Figure 12 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, which roughly captures the total energy consumed by the network.

As seen in Figures 11(a) and 11(c), keeping static replication points creates routing holes in the network, with 93 and 247 expired nodes after 30,000 and 50,000 cycles,

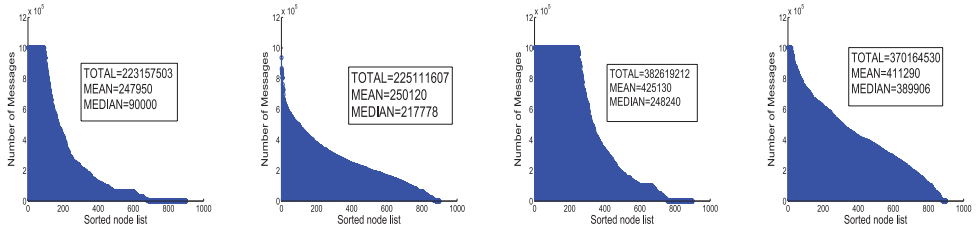
[2]We also ran experiments for $E_{th} = 100{,}000$, $E_{th} = 500{,}000$, and $E_{th} = 700{,}000$ messages, and in all of them, changing replicas clearly outperformed the static solution.

[3]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



(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. 11. Energy map from the number of messages sent and received by all nodes of the network (A = $300 \times 300\ m^2$, 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. 12. Distribution of the number of messages per node (A = $300 \times 300\ m^2$, 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).

respectively. The number of battery 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 (70,959 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 12, all nodes except 18 (2%) after 30,000 cycles and 15 (1.7%) after 50,000 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 30,000 cycles, decreasing to 160 (17.8%) nodes after 50,000 cycles. When considering the total energy consumed by the network, STARR-DCS uses just 0.8% more energy than the static one after 30,000 cycles. However, 3.3% extra energy is required by the static approach after 50,000 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

Table I. WSAN Lifetime ToW-Static vs. STARR-DCS for $\lambda_c > \lambda_p$

Lifetime Criteria	1^{st} *dead*	*1% dead*	*10% dead*	*25% dead*	*40% dead*	*10% cons+prod*	*25% cons+prod*	*Network disconnection*
ToW-Static (cycles)	2,619	7,328	29,086	47,830	63,230	31,668	47,984	70,952
STARR-DCS (cycles)	31,199	41,124	66,750	87,968	101,523	65,171	79,717	170,950
Improvement(%)	1,091%	461.2%	129.5%	83.9%	60.6%	105.8%	66.13%	140.9%

Table II. WSAN Lifetime ToW-Static vs. STARR-DCS for $\lambda_p > \lambda_c$

Lifetime Criteria	1^{st} *dead*	*1% dead*	*10% dead*	*25% dead*	*40% dead*	*10% cons+prod*	*25% cons+prod*	*Network disconnection*
ToW-Static (cycles)	8,341	10,518	53,140	84,625	109,070	51,491	82,987	105,310
STARR-DCS (cycles)	78,941	109,034	160,851	192,810	231,511	141,823	164,982	327,523
Improvement(%)	846.4%	936.6%	202.7%	127.8%	112.3%	175.4%	98.8%	211.0%

replicas over the time and using Random Replication extends the network's lifetime by a factor of $2\times$.

In order to demonstrate that the benefits exhibited by STARR-DCS when consumption-dominates-production also apply to the opposite case, production-dominates-consumption ($\lambda_p > \lambda_c$), we have repeated the same experiment using the same configuration, but now the network has 300 producers and 30 consumers that generate one event/query per simulation cycle. This makes ToW and STARR-DCS to use four replicas as in the previous case. Table II shows the network lifetime for both solutions. The lifetime extension shown by STARR-DCS is again huge as compared to the static ToW solution. It ranges between $2\times$ and $10\times$ depending on the chosen criteria.

Finally, Figure 13 shows how using a message threshold to trigger epoch changes compares to employing a fixed epoch duration as proposed [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 large (N = 5,000 nodes, A = 1000×1000 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 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. In addition, as it has been already mentioned, the proposed message threshold mechanism equalizes the energy consumption of replication nodes independently on the traffic load associated to a particular epoch. That is, epochs during high traffic period will be much shorter than epochs happening under low-traffic patterns (e.g., night).

5.3. Epoch Duration Analysis

We have demonstrated that changing replication nodes over the time leads to a huge improvement. In addition, we have also shown that using a message threshold to trigger epoch changes is a robust and fair mechanism (i.e., the cost of being replica does not depend on the traffic load in a particular epoch). However, the selection of a higher or shorter message threshold directly affects the epoch duration. For instance,

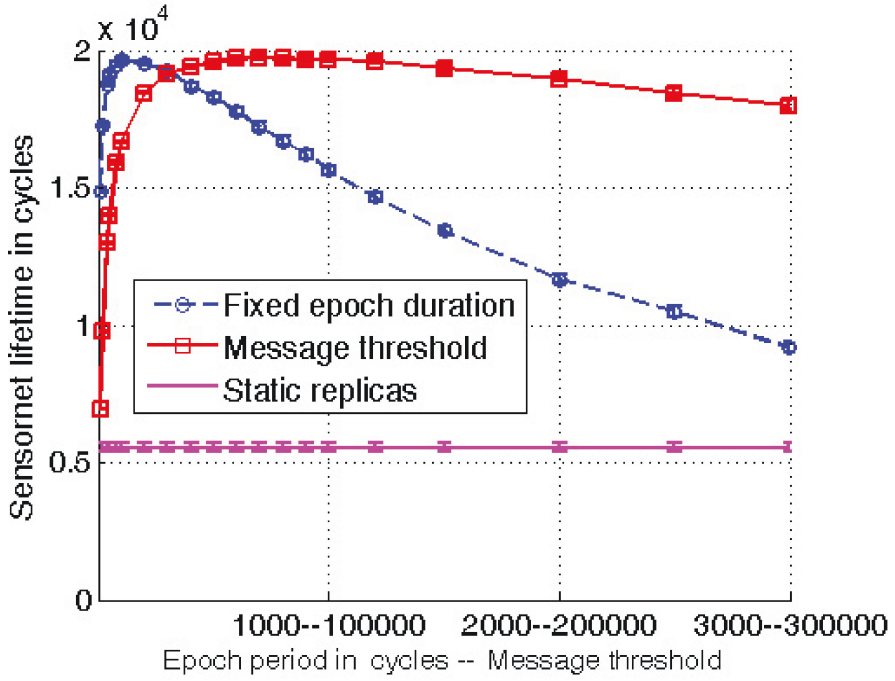


Fig. 13. WSAN lifetime comparison (A = 1000 × 1000 m^2, N = 5000 nodes, Tx = 50 m, L = 10 cycles, m = 5 applications, Battery = 10^6 messages). X axis refers to the cycles for changing the epoch in the fixed duration approach, or the message threshold.

if we consider an application with a constant rate for all producer events and consumer queries, all epochs' duration will be the same. Therefore the selected message threshold defines the epoch duration.

Thus the next question is whether it is better to use shorter or longer epochs. 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 stored data 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 energy consumption since nodes will keep being replicas for longer 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 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 (i.e., low message thresholds). Of course, the price of doing so is that a lot of extra energy is required for those frequent epoch changes. Thus, if 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 $A = 300 \times 300$ 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 one 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, 1,000, 5,000, 10,000, and 50,000 cycles per epoch. For each case, we ran the simulation for 50,000 cycles, resulting in a range of 5,000 epoch changes to none. For each epoch's 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 not having changes. However, as it was previously demonstrated the need for longer routing paths could have a larger impact on the 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. FI goes from 0 (lowest fairness) to 1 (highest fairness) and it is defined as

$$FI(x_1, \ldots\ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2},$$

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 14 shows the FI and the relative energy consumed for different epoch durations on a logarithmic scale.

As we expected, the lower the number of cycles per epoch (the more epoch changes), the better the network fairness, but the greater the overall energy required. However, there exists 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 5%, and the FI is good (> 0.75). This operational regime heavily depends on the value of L, which impacts the overhead associated with epoch changes. If L 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, 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 will have its own optimal operational regime.

6. STARR-DCS EVALUATION: TESTBED

Implementing WSAN solutions in real motes may be quite challenging. Usually theoretical proposals do not consider many practical issues that need to be taken into account when facing a real implementation. This claim is even stronger in the field of DCS where most of the proposals rely on assumptions that can be overtaken in a simulation but not in a real implementation. It must be noted that we could only find one previous implementation in the area of DCS, that is, the implementation of pathDCS [Ee et al. 2006].

We target two main goals in this section: (i) validate the feasibility of STARR-DCS in resource-constrained motes, and check the functionality of the protocols and algorithms

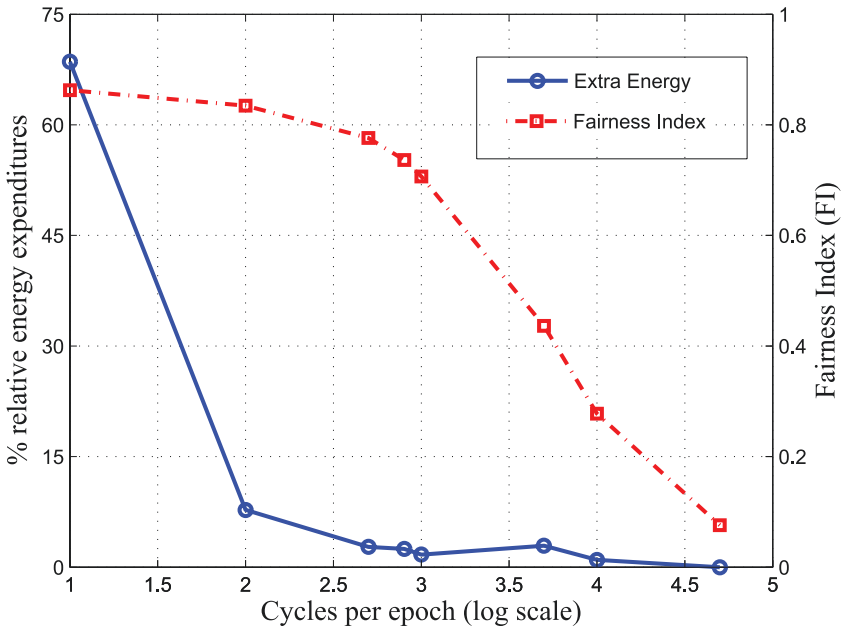


Fig. 14. Epoch duration analysis (A = 300 × 300 m^2, N = 900 nodes, Tx = 30 m, L = 10 cycles, simulation time = 50,000 cycles).

described in Section 2 (i.e., consumers and producers functionalities, replication-tree generation, change of replication nodes over the time, meta-information service, etc.); (ii) confirm that the results obtained from our simulation experiments also apply real WSN deployments that usually only include tens of nodes. To achieve these goals we have performed three different experiments on our STARR-DCS testbed.

Although STARR-DCS was designed from the very beginning taking into account practical issues, we still had to face several limitations during the implementation and testing phase. Next, we describe the main limitations to perform the planned experiments. The most relevant constraints were imposed by the limited operational capacity of Jennic motes (e.g., they do not operate with floating point numbers, they do not provide functions to perform dynamic memory allocation, etc). Moreover, the most important limitation to our work is that Jennic motes do not allow monitoring the battery level. This prevented the possibility of performing real measurements on battery depletion. Then, in order to validate simulation outcomes, we decided to use the number of messages sent and received (as we did in the simulations) as a rough estimation of the energy consumption, that is, the more messages a node sends/receives, the longer its radio transceiver is on (this is the element that dominates the energy consumption in a sensor node) and the sooner its battery is exhausted.

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

In all cases we have used a scenario with 20 motes located in four rows by five columns grid fashion emulating a 200 × 200 square meters network. Each node was programmed with its own coordinates and its neighbors' coordinates. We did not use full mesh connectivity, but a more irregular one in order to create longer communications

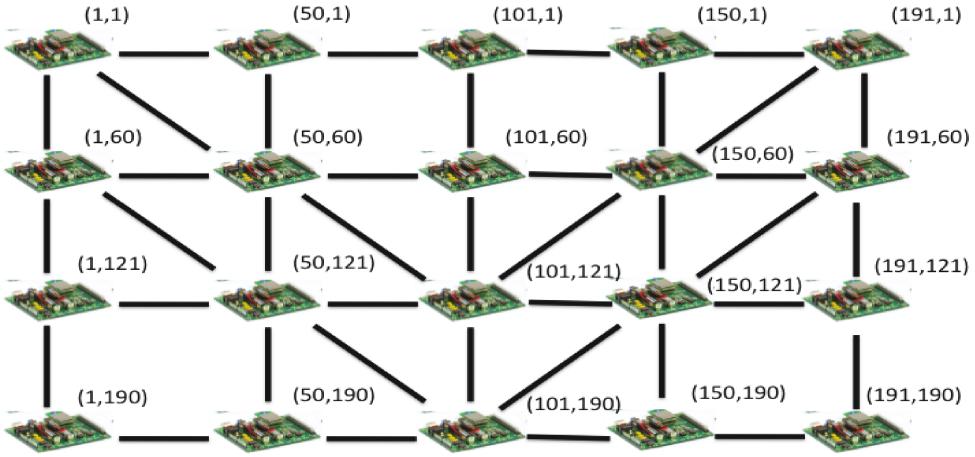


Fig. 15. STARR-DCS prototype using 20 motes emulating a 200 × 200 square meter network.

paths within the network. Figure 15 shows the testbed used to evaluate our framework implementation.

6.1. Optimal Number of Replicas

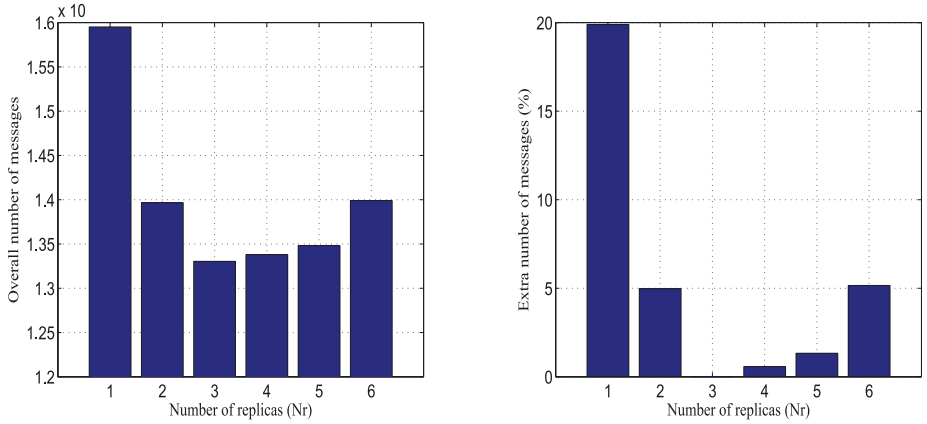
In this test, we always use the same number of replicas during the experiment, irrespective of the traffic statistics collected by the replication nodes. We tested different scenarios using from one to six replication nodes. We measure the total number of messages (sent and received) to account the traffic 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 three hours.

To avoid any bias in the traffic pattern and to better exploit the limited number of motes, we used all the nodes as producers and consumers, but we defined different consumption and production rates. The nodes generate a production event every 45 seconds and a consumption query every 15 seconds.[4] Following Eq. (1), the optimal number of replicas would be 4.2, thus mapping it to an integer, we obtain $N_r^* = 4$. Figure 16(a) shows the overall number of messages 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 three.

Figure 16(b) shows the extra traffic generated when using a number of replicas different than three. Then if four replicas are selected, which is the value chosen by our model, only 0.5% extra traffic is generated. The other close value to that obtained from the model is five replicas. In this case, the overhead traffic is 1%. However, if we choose those values far from the one provided by our model the extra traffic grows up to 5% for two and six replication nodes and up to 20% when a single replication node is selected. This demonstrates in a real testbed that using a single replication node as proposed in the seminal DCS work [Shenker et al. 2003] could generate a lot of extra traffic as compared to a multi-replication proposal.

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 provided by our model is leading to good results in terms of minimizing

[4]Producers are not synchronized when generating events. Therefore, it is very likely that each new consumer query finds new data stored in the replication nodes.



(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. 16. Evaluation of the optimal number of replicas.

network traffic (and reducing network energy consumption). This eliminates the uncertainty of how many replication nodes should be used for a particular application.

6.2. Balancing Energy Consumption

The second test we have conducted is to verify whether changing the replication nodes over time balances the energy consumption per node, even though it generates some overhead associated to epoch transitions. For that purpose we compare a static scenario to two dynamic ones that implement STARR-DCS, and have been 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 no node runs out of battery, and thus no network holes appear in the network for the static scenario as it happened in the simulation experiments (see Section 5). We remember that the appearance of routing holes in the static scenario leads to a larger traffic overhead than the one associated to epoch transitions in the dynamic scenario. Therefore, 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 three hours and all nodes were consumers and producers at the same time.[5] The production rate was one message every 45 seconds, and the consumption rate one query every 15 seconds, for all the nodes. As in the previous example, this leads to an optimal number of replicas $N_r^* = 4$. 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 one minute. It must be noted that usually the selected number of replicas was four, but sometimes a given node could take the responsibility of two hashed locations, so in that case, the actual number of replicas was three. Finally, we have used two different values for the message threshold in order to manage shorter and longer epochs. Those values were $E_{th} = 240$ and $E_{th} = 360$ messages, respectively. We remember that shorter

[5]Under this configuration, broadcasting would be a suitable solution, however we remind that the goal of this test is to understand whether STARR-DCS balances energy consumption when it is compared to a static solution.

Table III. Evaluation of Energy Distribution
Fairness Index (FI) of messages sent and received by network nodes and overhead generated by STARR-DCS as 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

epochs are expected to provide better fairness, but also higher overhead, since more messages are generated due to more frequent epoch transitions.

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 III shows the obtained 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 expected, the shorter the epoch (a lower message threshold), the better the energy distribution. That is the reason why the scenario with $E_{th} = 240$ messages presents a better FI than the one with $E_{th} = 360$ messages. On the other hand, changing the replication nodes over the time leads to increasing traffic overheads, which in our test was 10% for the STARR-DCS test with shorter epochs ($E_{th} = 240$ messages) and 7% in case of using longer epochs ($E_{th} = 360$ messages).

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 three-hour testbed means 30 epoch transitions (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.

6.3. 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 monitoring the remaining battery in a node, we emulate the battery lifetime as a limit for the maximum number of messages sent and received by the nodes.

For that purpose, we have evaluated a static scenario with random replicas and two scenarios implementing STARR-DCS with message threshold values of $E_{th} = 240$ and $E_{th} = 360$ messages, respectively. For each of these scenarios, we have measured the time when the first node reaches 5,000, 6,000, 7,000, 8,000, 9,000, and 10,000 messages in order to demonstrate that changing the replication set over the time produces an effective lifetime extension.

Table IV shows the effective time extension in percentage when STARR-DCS replaces a static solution. The first conclusion is that changing the replication set over the time reduces the load of the most saturated node in the network. This is translated into a longer period to reach the messages limit, which implies a longer time before running out of battery. In particular, by analyzing the scenario implementing STARR-DCS with $E_{th} = 240$ messages, we check that in all the cases evaluated the time extension is longer than 35%, and even goes above 50% when the messages limit is established in 9,000 messages. The time extension for $E_{th} = 360$ messages is reduced to values between 15% and 33%, depending on the different messages limits. This is happening

Table IV. Evaluation of Lifetime Extension when Adopting STARR-DCS Solution instead of a Static Approach
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. (%)
5,000	3,054	4,199	3,506	37%	15%
6,000	3,659	5,142	4,556	41%	25%
7,000	4,242	6,268	5,456	48%	29%
8,000	4,830	7,203	6,301	49%	30%
9,000	5,419	8,362	7,187	54%	33%
10,000	6,003	8,945	7,863	49%	31%

because a lower message threshold implies shorter epochs, thus a fairer distribution of energy consumed per node. This reduces the number of messages in the most loaded node 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 (i.e., node battery), the longer the lifetime difference between the static and the framework solutions. That means that if we were able to run very long tests (e.g., 1,000 epochs) like the ones evaluated in the simulations (see Section 5), the lifetime extension would be much higher, as suggested by the simulation results presented in Table I.

7. CONCLUSIONS AND FUTURE WORKS

This article has presented STARR-DCS, a framework that advances the state of the art in the field of data-centric storage. STARR-DCS is based on two main principles: (i) a random placement of several nodes serving as rendezvous nodes, and (ii) an equalization of the energy burdens across the network, by means of changing the set of rendezvous nodes over the time. On the one hand, Random Replication appears to be the most efficient replication algorithm in terms of minimizing the network traffic (i.e. overall energy consumption). In addition, it is computationally simpler and it is adaptable to any network shape. Moreover, Random Replication makes the proposed STARR-DCS framework suitable to work in networks without geographic information, which means an important advance in the field of DCS with multiple replication nodes. On the other hand, changing the replication nodes over the time allows to effectively extend the network lifetime between 60% and 10×, as demonstrated by our simulation study in large WSANs. Moreover, STARR-DCS implements a set of novel algorithms and protocols to address the complexity introduced by a dynamic WSAN environment. In order to test the feasibility of STARR-DCS we have successfully implemented it on resource-constrained commercial motes. Furthermore, that prototype has been used to perform several experiments that validate the main outcomes obtained from the large-scale simulation study in a small-scale scenario. Finally, our results conclude that DCS proposals using a single rendezvous node are highly inefficien in most scenarios.

As future research line, it may be interesting to look at mobile DCS networks, which present fairly different characteristics than the one addressed in this article for static WSANs. If we assume a full mobile network (all nodes are mobile) probably we do not need to create a dynamic framework since changing replicas may come with mobility alone. Given that a replication node is the closest one to a given position, that role will naturally change because nodes are not longer static, thus balancing the replication node role among them. However this could introduce quite a lot overhead, since every time a new node becomes the closest one to a particular replica location, it has to receive the information stored in the node that was the rendezvous node until that

moment. Therefore, in the case of mobile DCS networks, the research efforts should be directed to define algorithms that efficiently and unambiguously decide the proper rendezvous node(s) at a particular time and to design light protocols to access up to date information.

REFERENCES

AHN, J. AND KRISHNAMACHARI, B. 2006. Fundamental scaling laws for energy-efficient storage and querying in wireless sensor networks. In *Proceedings of MobiHoc'06*. ACM, New York, NY, 334–343. DOI:http://dx.doi.org/10.1145/1132905.1132942.

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. DOI:http://dx.doi.org/10.1109/TNET.2008.2009220.

BACCELLI, F. AND BORDENAVE, C. 2007. The radial spanning tree of a Poisson point process. *Ann. Appl. Probabi. 17*, 305.

BACCELLI, F., KLEIN, M., LEBOURGES, M., AND ZUYEV, S. 1997. Stochastic geometry and architecture of communication networks. *J. Telecommun. Syst. 7*, 1–3, 209–227.

BACCELLI, F. AND ZUYEV, S. 1996. Poisson-Voronoi spanning trees with applications to the optimization of communication networks. *Oper. Res. 47*, 4, 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. Info. 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 J. Select. Areas Commun. 22*, 6, 1130–1140.

CUEVAS, Á., 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. DOI:http://dx.doi.org/10.1145/1464420.1464425.

EE, C. T., RATNASAMY, S., AND SHENKER, S. 2006. Practical data-centric storage. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation (NSDI'06)*. USENIX Association, Berkeley, CA.

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.

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 *Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor System (DCOSS'08)*. Springer-Verlag, Berlin, 237–251. DOI:http://dx.doi.org/10.1007/978-3-540-69170-9_16.

KARP, B. AND KUNG, H. T. 2000. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of MobiCom'00*. ACM, New York, NY, 243–254. DOI:http://dx.doi.org/10.1145/345910.345953.

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. DOI:http://dx.doi.org/10.1016/j.compeleceng.2009.04.003.

MAYMOUNKOV, P. AND MAZIÈRES, D. 2002. Kademlia: A peer-to-peer information system based on the XOR metric. In *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems (IPTPS'01)*. Springer-Verlag, Berlin, 53–65.

MOLLER, J. 1994. *Lectures on Random Voronoi Tesselations*. Lecture Notes in Statistics, vol. 87, 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. DOI:http://dx.doi.org/10.1023/A:1024591915518.

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 *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*. ACM, New York, NY, 78–87. DOI:http://dx.doi.org/10.1145/570738.570750.

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. DOI:http://dx.doi.org/10.1145/774763.774785.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM Conference*.

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 *Proceeding of the IEEE Wireless Communications and Networking Conference (WCNC'06)*. IEEE, Los Alamitos, CA, 2168–2174. DOI:http://dx.doi.org/10.1109/WCNC.2006.1696632.

Received November 2011; revised September 2012, February 2013; accepted March 2013