Resilient Data-Centric Storage in Wireless Ad-Hoc Sensor Networks

Abhishek Ghose, Jens Grossklags, and John Chuang

University of California at Berkeley
aghose@eecs.berkeley.edu, {jensg,chuang}@sims.berkeley.edu

Abstract. Wireless sensor networks will be used in a wide range of challenging applications where numerous sensor nodes are linked to monitor and report distributed event occurrences. In contrast to traditional communication networks, the single major resource constraint in sensor networks is power, due to the limited battery life of sensor devices. It has been shown that data-centric methodologies can be used to solve this problem efficiently. In data-centric storage, a recently proposed data dissemination framework, all event data is stored by type at designated nodes in the network and can later be retrieved by distributed mobile access points in the network. In this paper we propose Resilient Data-Centric Storage (R-DCS) as a method to achieve scalability and resilience by replicating data at strategic locations in the sensor network. Through analytical results and simulations, we show that this scheme leads to significant energy savings in reasonably large-sized networks and scales well with increasing node-density and query rate. We also show that R-DCS realizes graceful performance degradation in the presence of clustered as well as isolated node failures, hence making the sensornet data robust.

1 Introduction

Wireless sensor networks have emerged as a promising solution for a large number of monitoring applications, in fields as diverse as climatic monitoring, tactical surveillance, and earthquake detection. With improvements in sensor technology, it has become possible to build small sensor devices with relatively high computational power at low costs.

In most communication networks, naming of nodes for low-level communication leverages topological information. An example of this is the Internet (*point-to-point communication model*) where IP addresses are assigned to each node, and these serve as unique node identifiers in IP routing. Such a naming scheme is not very efficient in a sensor network scenario, since the identity of individual sensor nodes is not as important as the data associated with them. *Data-centric* models have been proposed for sensor networks, in which the sensor data itself (as contrasted to sensor nodes) is *named*, based on attributes such as event-type or geographic location. In particular, data-centric routing [5] and data-centric storage [9] have been shown to be energy-efficient in a sensor network scenario.

M.-S. Chen et al. (Eds.): MDM 2003, LNCS 2574, pp. 45–62, 2003.

Ratnasamy *et al.* have proposed data-centric storage (DCS) [7,9] as a data-dissemination paradigm for sensor networks. In DCS, data is stored, according to event type, at corresponding sensornet nodes. All data of a certain event type (*e.g.*, Temperature measurements) is stored at the same node. A significant benefit of DCS is that queries for data of a certain type can be sent directly to the node storing data of that type, rather than flooding the queries throughout the network (unlike data-centric routing proposals [4,5]). DCS is based on the low-level routing functionality provided by the GPSR geographic routing algorithm [6], and on distributed hash-table functionality provided by peer-to-peer lookup algorithms such as CAN [8] and Chord [10]. It is shown in [7] and [9] that DCS offers reduced total network load and peak (hotspot) network usage.

In this paper, we propose replication of control and data information in a DCS framework as the primary mechanism for reducing the data retrieval traffic and increasing resilience to node failures. We propose the storage of data of a particular type at one of several *replica nodes* in the network assigned to this type, and storage of control and summary information pertaining to this type at geographically distributed *monitor nodes* in the network. By increasing the number of nodes where data can be stored for each event-type, as well as maintaining summary and control information at several nodes, we decrease both (1) the average cost of storing data and (2) the average cost of querying data. Our preliminary results suggest that this scheme, which we call Resilient Data-Centric Storage (R-DCS), outperforms existing schemes in terms of scaling to a large number of nodes and a large number of queries. We also show that, in the case where nodes in the sensor network are unreliable and experience random failures, our scheme does not experience a dramatic increase in the number of messages sent as the node failure rate is increased. R-DCS also maintains a high query success rate in this scenario. Hence our scheme realizes graceful performance degradation in the presence of node failures.

The rest of the paper is organized as follows. In Section 2, we describe some basic concepts of sensor networks and give a brief overview of DCS. In Section 3, we propose intelligent replication in sensor networks for resilience and scalability, building upon the DCS framework. We analyze the cost structure of R-DCS analytically in Section 4. Section 5 presents simulation results showing that R-DCS can lead to significant performance improvements in certain scenarios. Finally, we present our conclusions in Section 6.

2 Background and Related Work

In this section we describe basic concepts in the domain of sensor networks, describe data organization in a typical sensor network, and give a brief overview of DCS [9].

2.1 Overview

Wireless sensor networks have certain unique features which must be accounted for in any data dissemination methodology designed for such networks. Sensor

devices built using state-of-the-art technology have significantly higher processing capabilities and storage capabilities than available bandwidth [4] (this is in contrast to wired networks, where an explosion of available bandwidth has led to a drastic reduction in its relative cost). The reason for this difference is that sensors typically have limited battery-life. Hence they must use low-power (and consequently, low-bandwidth) wireless communication techniques to conserve battery power. This encourages the use of computational techniques to reduce the total communication overhead in the network.

The gateway through which sensor networks communicate with the external world (*e.g.*, a monitoring terminal or the Internet) is called an *access point*. We use the term *access path* to refer to the set of data paths from the sensor nodes to the access point. In a typical scenario, we can expect these access points to have higher communication load than other sensornet nodes. In a high-traffic scenario, such access points can become a bottleneck point in the sensornet (*hotspot*). An essential design requirement, then, is to minimize the peak amount of traffic flowing through these access points. This issue is addressed in Section 3.4.

2.2 Data Organization in Sensor Networks

Prior research [1,4] has shown that the tight energy constraints of wireless sensor networks can be more efficiently achieved by using an *attribute-based* naming system than by commonly used topological naming schemes (*e.g.*, IP). Such attributes could be pre-defined to reduce the overhead during actual communication. For example, we could classify all sensor data in an environmental sensing network as being of types "temperature", "pressure" and "humidity" and name all such data by including these pre-defined *event attributes* in the data itself.

At the lowest level, when an event occurs the sensors record and store the event data locally, and *name* this data based on its attributes. The low-level output from sensors (*observations*) is *named* based on the attributes of the associated data. This data can be handled in a number of different ways. Three canonical approaches [9] are considered: *External Storage (ES)* in which all event data is stored at an external storage point for processing, *Local Storage (LS)* in which all event information is stored locally (at the detecting node) and *Data-Centric Storage (DCS)* in which all event data is stored by event-type within the sensornet at designated nodes. These three methods involve substantially different assumptions and cost-benefit tradeoffs, which we analyze in Section 4.

Queries are used to retrieve event information from the sensornet. It is important to consider the ratio of query traffic to event-detection traffic while designing a sensor network. Each of the canonical approaches described above (as well as our approach: R-DCS) has different relative costs for query and event traffic. Hence, depending on the function of a sensor network, one of the above *canonical approaches* might be more useful than the others.

2.3 Data-Centric Storage

In this section we describe the concept of DCS [9] and the mechanisms required to support it. DCS uses a distributed hash-table (DHT) and offers the following

interface: (1) the **Put(dataName, dataValue)** primitive to store the value of the data corresponding to a certain event at the sensornet node corresponding to the dataName (which serves as the *key* in the DHT and is typically based on the *event-type*). The name of the data is typically based on the relevant *event-type*. (2) the **Get(dataName)** primitive to retrieve the value of the data stored at the node corresponding to the given dataName. DCS uses the GPSR [6] geographic routing algorithm for low-level routing. It then builds a DHT [8,10] on top of GPSR.

DHT over GPSR The central idea in using a DHT is to hash the *name* of a certain event to a key (*dataName*) which is a location somewhere within the boundaries of the sensornet. The *put(dataName, dataValue)* primitive sends a packet with the given payload into the sensornet which is routed towards the location *dataName*. The *get(dataName)* primitive is routed to the node closest to the *dataName* location, which then transmits a packet to the node originating the query with the corresponding data. In a sensornet with completely stationary and reliable nodes, this approach is sufficient.

DCS Extensions In order to make DCS resilient to node failures and mobility, the authors [7,9] have proposed certain extensions to their basic scheme. The *storage node* for an event type periodically routes a *refresh* message to all nodes which had transmitted event-data to this node. Regular GPSR routing returns these refresh messages to the storage node along the network perimeter. In the intermittent time interval, the nodes in the sensor network could be displaced from their original locations. If a new node is closer to the *location of the original storage node* than the original storage node itself, this new node will become the storage node. Timer based algorithms ensure that in case a storage node dies in this fashion, the new storage node automatically starts generating refresh messages. This process is called the Perimeter Refresh Protocol [7] and is used to accomplish replication of (key,value) pairs and their consistent placement at the appropriate home nodes when the network topology changes.

In [7] Ratnasamy *et al.* have proposed a scheme called Structured Replication in DCS (SR-DCS) to achieve load-balancing in the network. SR-DCS uses a hierarchical decomposition of the key space and associates each event-type e with a hierarchy depth d. It hashes each event-type to a *root* location. It then computes $4^d - 1$ *images* of *root*. When an event occurs, it is stored at the closest image node. Queries are routed to all image nodes, starting at the *root* and continuing through the hierarchy. It has been shown through simulations that SR-DCS significantly improves the scalability of DCS, and is useful for frequently detected events. It must be noted, however, that SR-DCS does not involve actual *replication* of data. It only stores *one* copy of any event-data at the closest image node. If all nodes in a certain location fail simultaneously (clustered failures), SR-DCS might not be able to recover the data stored at these nodes. The *root* node is a single point-of-failure in the sense that if the *root* for event-type e fails,

one might not be able to issue any queries for event-data corresponding to this type. These extensions are described in detail in [7].

3 Resilient Data-Centric Storage

In this section we describe our extensions to DCS which achieve the following: (a) Minimize query-retrieval traffic (hence saving energy consumption) (b) Increase data availability, ensuring that event information is not lost even with multiple node failures. The original version of DCS [9] has all events of the same type stored in one sensornet node. It is evident that if there are too many events of a particular type, then this storage node will become a bottleneck point (*hotspot*) in the network. Our data-dissemination scheme, Resilient Data-Centric Storage (R-DCS), overcomes these issues by a two-level replication strategy (*control* and *data*). We first describe the architecture of R-DCS and then outline its operational details.

3.1 Architecture

In R-DCS, we partition the coordinate space of the sensornet field into Z zones. We denote the set of available zones as $z_j : j = 1, \cdots, Z$. This zoning could be done on the basis of geographical boundaries, as shown in Figure 1. These zones can contain sensor nodes operating in three possible modes:

- **Monitor Mode**: Each zone has one monitor node for each event-type. The monitor node stores and exchanges information in the form of a *monitoring map* for each event-type. The monitoring map includes control and summary information in the following fields:
  - *List of zones containing replica nodes* (for forwarding event data and queries).
  - *List of zones containing monitor nodes* (for facilitating map exchange).
  - *Event summaries* (for facilitating summary-mode queries). The exact nature of event summaries depends on the event-type. For example, in a sensornet designed for temperature monitoring, the summary information could contain the number of events detected and the average temperature reading for each zone.
  - *Bloom filters* (for enabling attribute-based queries). Event-data is organized in the form of a set of attributes and their values. In the temperature monitoring case, for example, these attributes could be Event-time and Temperature. A user might want to access all temperature readings between 30 and 40. Bloom filters [2,3] offer an efficient way to support attribute-based queries. These are described in detail in Section 3.2. Note that this field is required only if we want to support attribute-based queries.
- **Replica Mode**: Each zone has at most one replica node for each event-type. The replica node, if present, is always the same as the monitor node. In addition to performing the functions of a monitor node, the replica node actually stores event-data for the given event-type.

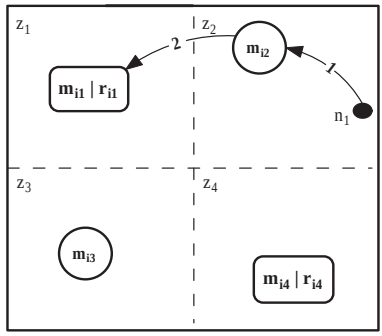

Fig. 1. Event Storage in RDCS

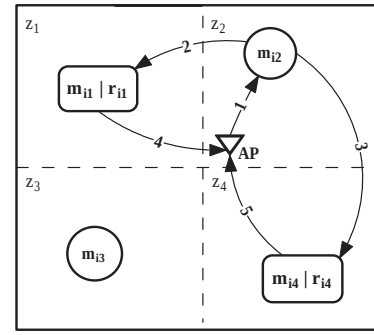

Fig. 2. Querying in R-DCS (*list*)

– **Normal Mode**: All nodes which are not *monitor* or *replica* operate in this mode. A normal node may originate or forward (*i.e. route*) event-data, but is not involved in storing any event data or control information.

Let E denote the number of event-types in the sensornet. Let M_i be the total number of monitor nodes for each event type e_i. Let R_i be the number of replica nodes for each event-type e_i. The following system constraints must be satisfied for each $i = 1, \cdots, E$.

1. $R_i \leq M_i \leq Z$ - This holds because each zone may have at most one replica node and one monitor node, and all replica nodes are monitor nodes as well. Under normal operations without clustered node failures (a majority of nodes failing in one zone), there will be one monitor node per zone: $M_i = Z$.
2. $R_i \geq 1$ - There must be at least one replica node in the network, since all event-data for each event-type is stored at the respective replica nodes.

For our DHT, we use a hash function H which is a function of the event-type e_i and the zone z_j. If event-type e_i in zone z_j hashes to a location $(x_{ij}, y_{ij}) \equiv H(e_i, z_j)$, then a sensor node m_{ij} geographically closest to (x_{ij}, y_{ij}) is the monitor node for event-type i within zone j. Depending on local decision rules (described in Section 3.2), this monitor node may also serve as a replica node r_{ij}. For load balancing, it is desirable that the function $H(e_i, z_j)$ be chosen such that for each zone z_j, different event-types e_i hash to distinct nodes.

3.2 Operational Procedures

In this section, we describe the methodology for performing common sensornet operations such as data storage and queries.

Event Storage A sensing node (situated in zone j) sends an event of type e_i to the monitor node in the same zone m_{ij}. If this monitor node is also the replica node r_{ij}, then the event-data is stored at r_{ij}. If not, the data is forwarded

to the *closest* replica node for this event-type. The closest replica node can be determined from the information in the *list of replica nodes* field of the local monitoring map. The target replica node stores the event-data and updates its local copy of the monitoring map. This operation is illustrated in Figure 1.

Event Query We envision three types of queries in an R-DCS system - *summary*, *list* or *attribute-based*. These are now described in detail, along with mechanisms to support these in R-DCS.

- **List**: A list query for an event-type is a request for *all* stored data for events of this type. A querying node in zone z_j sends the query for event-type e_i to the local monitor (and possibly replica) node m_{ij}. The monitor node then duplicates the query and forwards it to all other active replica nodes $r_{ix} : x = 1, \cdots, Z$ in the sensornet. All active replica nodes reply directly to the querying node with event-data. This operation is illustrated in Figure 2.
- **Summary**: As the name suggests, the querying node requests a summary of event information for an event-type. A querying node in zone z_j sends the query for event-type e_i to the local monitor node m_{ij}. The monitor node responds with the *event summary* information from the local monitoring map. This is illustrated in Figure 3.
- **Attribute-based**: An attribute-based query requests data for all events which match certain constraints on their attribute values. A querying node in zone z_j sends the query for event-type e_i to the local monitor (and possibly replica) node m_{ij}. The monitor node then duplicates the query and forwards it to all other active replica nodes in the sensornet with Bloom filter *matches*. Bloom filters are explained in some detail in Section 3.2. All active replica nodes reply directly to the querying node with event-data.

Periodic Update and Exchange of Monitoring Map When an event of type e_i occurs in zone j, it updates the local monitoring map in m_{ij}. However for

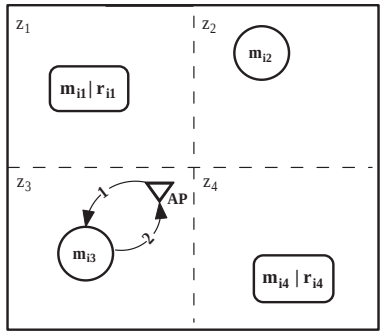

Fig. 3. Querying in R-DCS (*summary*)

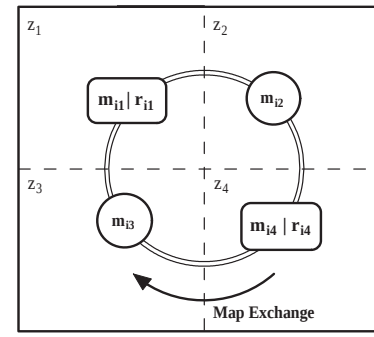

Fig. 4. Logical Ring in R-DCS

global consistency of information such as the number of replica nodes for type e_i and event-summary information, these monitoring maps must be exchanged between the respective monitor nodes at periodic intervals. For this purpose, all active monitor nodes for a type e_i form a logical ring as shown in Figure 4. Each zone has two adjacent zones. When a monitor node receives a new map, it adds its own local updates (based on events received since the last map update) and forwards it to the next monitor node.

Switching between Modes A node in zone j switches from *normal* mode to *monitor* mode (for event-type i) when it becomes the node closest to the location $(x_{ij}, y_{ij}) \equiv H(e_i, z_j)$. If due to node mobility, a new node $m_{ij}\prime$ becomes closer to this location, then the monitoring map is handed over from m_{ij} to $m_{ij}\prime$, and $m_{ij}\prime$ becomes the new monitoring node in zone j. $m_{ij}\prime$ also becomes a replica node $r_{ij}\prime$ if $m_{ij} \equiv r_{ij}$ was a replica node, and the relevant stored data is handed over from r_{ij} to $r_{ij}\prime$.

A node switches from *monitor* mode to *replica* mode and vice-versa based on certain local criteria such as event storage and query traffic loads, as well as residual energy in the node. In R-DCS, every monitor/replica node logs all its storage and query traffic loads for a certain window of time $(t - \tau, t)$, as well as its residual energy at time t, and uses a composite of this information to determine the current *activity coefficient* $A_{ij}(t)$ of a monitor/replica node for event-type e_i in zone j. The residual energy term accounts for the fact that sensornet nodes have limited energy, and hence replica nodes try to conserve energy (by becoming a monitor node) when their residual energy is running *low*.

We define two threshold values for the activity coefficient for event-type e_i: a lower threshold $\lambda_{i(r \to m)}$ and an upper threshold $\lambda_{i(m \to r)}$ (with a buffer zone in between to account for hysteresis effects). A monitor-node m_{ij} switches to replica mode when it's current activity coefficient exceeds the upper threshold, *i.e.*, $A_{ij}(t) > \lambda_{i(m \to r)}$). Conversely, a replica-node r_{ij} switches to monitor-mode when it's current activity coefficient goes below the lower threshold. The only constraint is that there must be at least one replica node in the network. Further, any replica node switching to monitor mode must handoff its stored data to one of the other replica nodes. The values of $\lambda_{i(r \to m)}$ and $\lambda_{i(m \to r)}$ are determined based on the relative costs of storage, queries and of conserving residual energy.

Handling Node Failure In the above discussion we assumed that all nodes are stable and able to continuously route, monitor or store data. Consider the more realistic case, when nodes fail with a certain failure probability f. We will discuss the effect of node failure and mechanisms to solve this problem in the following sections.

- **Failure of Monitor and Replica Nodes** When a monitor node in a certain zone fails, we can use PRP described in [7] and in Section 2.3 to find/elect an alternate monitor node for this zone. PRP also ensures that the new monitor node has a copy of the monitoring map through local refresh mechanisms. If

this failed node is also a replica node, the event data stored in it might be lost. However using PRP, it is possible to recover at least part of this data through local refreshes.

- **Clustered Node Failures** Under normal operating conditions, there is one monitor node per zone *i.e.*, $M_i = Z$ for each event-type e_i. These nodes form a logical ring for the purpose of updating their maps. However, if all nodes in a particular zone fail, it results in a break of the logical ring of monitor nodes. Note that in every map-update cycle, the monitor nodes *mark* their presence by updating the *List of monitoring nodes* field of the monitoring map. By checking this field, it is possible to discover which zone(s) have experienced clustered node failures, and to route around these zones to reconstruct a logical ring of $M_i < Z$ monitor nodes. We can employ different techniques (*e.g.*, beaconing mechanisms) to recover from such failures.

Bloom Filter-Based Attribute Matching Bloom filters [2,3] can be used to support attribute-based queries of the event-list stored at each of the replica nodes for an event type. It is a method to represent a set of elements as a compact summary supporting membership queries. Consider a set of v elements to be represented using an u-bit long vector with all entries initially set to zero. A set of k independent hash functions $h_1, \cdots, h_k$ is chosen where each function has a range of 1 to u. For each element x to be represented, the bits at positions $h_1(x), \cdots, h_k(x)$ in the bit vector are set to 1. A bit may be set to 1 multiple times. This bit vector serves as a summary. To find out if an element y is in the event-list, we check the positions $h_1(y), \cdots, h_k(y)$ in the bit vector. If they are all set to 1, then we can infer that y is in the event-list, though there is a (small) probability of being wrong (false positive). The critical feature of a Bloom filter is that the probability of false positives decreases exponentially with u, if the number of hash functions k is chosen optimally. It is shown in [3] that the probability of a false positive is given by $(1 - (1 - \frac{1}{u})^{kv})^k$, which gives the optimal value of k as $\frac{u}{v} \ln 2$.

We now explain how this concept can be applied to the problem of attribute-based queries in a sensor network. Consider the temperature monitoring example. The Bloom filter can be used to represent compactly which replica nodes contain temperature readings in certain temperature ranges. Suppose a query node sends the query "Get all event data for temperature readings between 30 and 40" to the local monitor node. By Bloom filter-based matching, we can obtain a list of replica nodes containing relevant temperature readings. The local monitor node can then forward the query to this restricted set of replica nodes. Note that the usage of Bloom filters is optional. It is appropriate only when there are a large number of replica nodes, making flooding of all replica nodes expensive.

3.3 Performance Measures

We need to specify relevant performance measures to quantify the benefits of our DCS mechanisms. As mentioned before, data dissemination algorithms should

seek to minimize communication in order to extend overall system lifetime. For consistency and for the sake of easy comparison, we use the same metrics to quantify communication overhead as Ratnasamy *et al.* [7,9]. These metrics (assuming that all packets have the same size) are:

- **Total Usage:** Total Number of Packets sent in the sensornet.
- **Hotspot Usage:** Maximal Number of Packets sent by any particular sensornet node.

As previously mentioned, when an event of a particular type occurs at any sensornet node, it transmits this event data to the closest monitor node, which then forwards it to the closest replica node for storage of events of that type. If the data storage operation fails (this could occur if the destination node is unreachable or malfunctioning), this data is retransmitted to one of the other R_i replica nodes. This process is iterated until the data is successfully stored. If all the R_i replica nodes for event type i fail simultaneously, then an attempt to store or retrieve data of this type will *fail*. To measure the success rate of queries for event data, we use the metric **Query Success Rate**, which is the mean percentage of queries (averaged over all Event Types) which return a successful response. *Total Usage*, as a function of node failure rate f, can also be used as a resilience metric. Total Usage is expected to increase as the node failure rate increases (due to more retransmissions). However, a drastic increase in Total Usage at higher node failure rates would imply that our scheme is not robust in the presence of node failures. Hence R-DCS aims to achieve graceful degradation (with respect to Total Usage) in system performance, as f is increased.

4 Analytical Results

We now derive analytical expressions for energy costs and savings obtainable with R-DCS. We consider a sensornet with N nodes equipped to detect E event types. Let D_{total} denote the total number of events detected, Q the number of queries issued, D_q the number of events detected for these queries, R be the number of replica nodes, M the number of monitor nodes and ω the frequency of monitoring map updates.

4.1 Energy Savings

The communication costs of the three canonical methods [9] and of R-DCS can be estimated using the fact that the asymptotic cost of a flood is $O(N)$ and that of direct routing from one random node to another (using GPSR) is $O(\sqrt{N})$. For R-DCS, we estimate the total message count (*total usage* T) and the number of messages at the busiest point (*hotspot usage* H). We reproduce from [9] the corresponding expressions for each of the aforementioned *canonical methods* [9] for completeness. The formulae presented give the *expected values* of these quantities. We consider a scenario in which we have one access point. The results can be easily generalized for multiple access points. We assume that the

message counts at the access point is a good estimate of hotspot usage, since it is likely to be the busiest area of the network.

1. **External Storage**:
   The cost of storing each event (sending to external store) is $O(\sqrt{N})$. There is no cost for queries since event information is already external.

$$T = D_{total}\sqrt{N} \qquad H = D_{total} \tag{1}$$

2. **Local Storage**:
   Storing event information locally incurs no cost. Queries are flooded to all nodes at a cost of $O(N)$. Responses are routed back to the query-source at a cost of $O(\sqrt{N})$.

$$T = QN + D_q\sqrt{N} \qquad H = Q + D_q \tag{2}$$

3. **Data-Centric Storage**:
   Storing event information at a sensornet node depending on the event-type incurs a cost of $O(\sqrt{N})$. Queries for a particular event-type are routed to the storage node, which returns a response, both at a cost of $O(\sqrt{N})$. Total usage depends on whether a full-*list*ing of events (l) is required (uses one packet for every instance of an event for each event-type) or whether a *summary* (s) is sufficient (uses only one packet for each event-type).

$$\begin{array}{lll} T = D_{total}\sqrt{N} + Q\sqrt{N} + D_q\sqrt{N} & H = Q + D_q & \text{(l)} \\ T = D_{total}\sqrt{N} + Q\sqrt{N} + Q\sqrt{N} & H = 2Q & \text{(s)} \end{array} \tag{3}$$

   In the case of Structured Replication with DCS (SR-DCS), the costs of storage is decreased whereas the cost of queries is increased (compared to plain DCS). For a hierarchy-depth d, the storage cost for a single event decreases from $O(\sqrt{N})$ to $O(\sqrt{N})/2^d$. The cost of routing a single query through the complete image-node hierarchy increases from $O(\sqrt{N})$ (for DCS) to $O(2^d\sqrt{N})$. Total usage is given by:

$$\begin{array}{lll} T = D_{total}\sqrt{N}/2^d + Q2^d\sqrt{N} + D_q 2^d\sqrt{N} & H = Q + D_q & \text{(l)} \\ T = D_{total}\sqrt{N}/2^d + Q2^d\sqrt{N} + D_q 2^d\sqrt{N} & H = 2Q & \text{(s)} \end{array} \tag{4}$$

4. **Resilient Data-Centric Storage**:
   Since event information is stored at the node closest to the event-location, storage costs are reduced from $O(\sqrt{N})$ to $O(\sqrt{N/R})$. Queries of a particular event-type are routed to the closest monitor node at a cost of $O(\sqrt{N/M})$. In the *summary* case described above, this monitor node can directly return a response (based on the monitor-maps it receives from the other nodes) at a cost of $O(\sqrt{N/M})$. In the *list* case, the query must be forwarded to all R replica nodes (at a cost of $O(R\sqrt{N})$), and all of them return responses, at a cost of $O(\sqrt{N})$. Let the frequency of monitoring map updates be ω. The cost of exchanging monitor maps is $O(2\omega\sqrt{NM})$ for each event-type.

$$
\begin{array}{lll}
T = D_{total}\sqrt{N/R} + Q(\sqrt{N/M} + R\sqrt{N}) + D_q\sqrt{N} + 2\omega E\sqrt{NM} & \text{(l)} & \\
T = D_{total}\sqrt{N/R} + Q\sqrt{N/M} + Q\sqrt{N/M} + 2\omega E\sqrt{NM} & \text{(s)} & (5) \\
H = Q + D_q \quad \text{(l)} \qquad H = 2Q \quad \text{(s)} &&
\end{array}
$$

From the above formulae, we can draw a number of conclusions. If N is increased, other parameters constant, the local storage method incurs the highest total message count. If $D_q \gg Q$ and events are summarized, DCS/R-DCS have the lowest hotspot usage. If $D_{total} \gg D_q$, ES has significantly higher hotspot usage than the other schemes. In general, DCS/R-DCS are preferable in cases where (1) N is large. (2) $D_{total} \gg D_q \gg Q$ (Many detected events and not all event types queried). However, LS may be preferable if the number of events is large compared to system size ($D_{total} > Q\sqrt{N}$) and event *lists* are used.

Comparing the total message counts for R-DCS and DCS/SR-DCS (R in R-DCS is equivalent to 4^d in SR-DCS), we see that R-DCS will outperform DCS in the *summarized* case, for typical values of the scenario parameters. Considering $N = 10000$, $E = 100$, $D_{total} = 10000$, $Q = 50$, $D_q = 5000$, $R = 4$, $M = 16$ and $\omega = 0.1$ we get $T = 1020000$ for DCS, $T = 540000$ for SR-DCS and $T = 513000$ for R-DCS. In the *list* case, the $T = 1505000$ for DCS, $T = 1510000$ for SR-DCS and $T = 1029250$ for R-DCS. Hence we see that R-DCS could provide significant energy savings for both the *summarized* and the *list* case. R-DCS presents an intermediate solution between LS (free storage, expensive queries) and DCS (both at moderate cost). It can be expected to give good performance when $D_q \gg Q$.

4.2 Increased Resilience

An obvious benefit of R-DCS over DCS is the increased resilience to node failures. Having M monitor nodes and R replica nodes for each event-type ensures that all information corresponding to an instance of a particular event-type is not lost when one node fails. Further, since these nodes exchange and replicate *monitoring maps*, it is extremely unlikely that the control and event-summary information is lost.

Let us consider a situation in which nodes in the sensor network are unreliable, so that they are *on* (*i.e.*, functioning correctly) with probability $1 - f$, and *off* (*i.e.*, malfunctioning) with probability f. Here $0 \leq f \leq 1$. $f = 0$ corresponds to the case of perfectly reliable nodes. As in the previous section, we approximate the communication costs (T) for R-DCS. Local refresh mechanisms like PRP [7] are used to ensure that there is one monitor node in every zone, except in the case of clustered node failures. *Normal* nodes in any zone j communicate with the local monitor (and replica if it exists) nodes m_{ij} (r_{ij}) for data of event-type e_i. In any message, the monitor (replica) node includes a list of *viable* destinations in the header. For example, when a source monitor node m_{ij} needs to store its event data at one of the R_i replica nodes for type i event data, it constructs the destination list by computing its distances to all the type i replica nodes listed in the local monitoring map and then ordering the R_i replica nodes according to their distances from the source. Hence the destination of first choice is the

closest replica node, the first *alternate* destination is the second-closest replica node, and so on. m_{ij} then routes the message to each one of the R_i possible destinations in its list, in order of preference. If all R_i destinations are exhausted without success, the message is dropped.

In Resilient Data-Centric Storage, a single message has a cost $O(\sqrt{N})$, when the destination is *on* (with probability $1-f$). The probability of the destination being *off* is f, in which case the message is sent to the first *alternate* destination. In the second transmission, with probability $1-f$ the message is successfully sent at a total cost of $2O(\sqrt{N})$ [which consists of a cost $O(\sqrt{N})$ for the first failed message and a cost $O(\sqrt{N})$ for the second successful message] or it fails again with probability f and so on. This iterative process is represented by Equation 6.

$$\begin{aligned} C &= (1-f)\sqrt{N} + f((1-f)2\sqrt{N} + f((1-f)3\sqrt{N} + \ldots)) \quad R \text{ terms, } R \geq 2 \\ &= \sqrt{N} C_{Rf} \end{aligned} \tag{6}$$

$$C_{Rf} = \frac{1-(R+1)f^R + Rf^{R+1}}{1-f} \tag{7}$$

We can now give the expressions for T in this case.

$$\begin{aligned} T &= D_{total}\sqrt{N/R}C_{Rf} + Q(\sqrt{N/M} + R\sqrt{N})C_{Rf} + D_q\sqrt{N} + 2\omega E\sqrt{NM} \quad \text{(l)} \\ T &= D_{total}\sqrt{N/R}C_{Rf} + Q\sqrt{N/M}C_{Rf} + Q\sqrt{N/M}C_{Rf} + 2\omega E\sqrt{NM} \quad \text{(s)} \end{aligned} \tag{8}$$

Let us denote the probability of successful storage of data corresponding to a particular event instance as P_s (here the subscript i is omitted, but it must be noted that we are considering a particular event-type e_i). Further let us denote the probability of *clustered* node failure as f_c, in which a majority of the nodes in a particular zone malfunction. As described earlier, this operation succeeds if all R replica nodes for the corresponding event type do not fail simultaneously. The probability of simultaneous failure of R replica nodes (assuming these failures are independent) is f^R. Clearly, $P_s = 1 - f^R$. For a *summary* query to be successful, two conditions have to be satisfied: (1) The event data being queried must have been successfully stored (which occurs with probability P_s). (2) The local monitor node must be alive storing this summary data. Since we assume that local refresh mechanisms like PRP [7] ensure the existence of a monitor node except in the case of clustered node failures, this monitor nodes is alive with probability $1 - f_c$. Hence the mean query success rate Q_s is given by

$$Q_s = (1-f^R)(1-f_c) \qquad \text{(s)} \tag{9}$$

For a *list* query to be successful, the second condition is modified to all monitor nodes being alive. This occurs with probability $(1-f_c)^M$. Hence the mean query success rate is given by

$$Q_s = (1-f^R)(1-f_c)^M \qquad \text{(l)} \tag{10}$$

From equations 7 and 8, we can see that if f is increased, other parameters constant, then T increases. This should be obvious intuitively, since a greater number of unreliable nodes imply more retransmissions. Also from equations 9 and 10, it is clear that if R is increased keeping f constant, Q_s increases. Similarly if f is increased, Q_s decreases (for the same value of R). These variations are further investigated in Section 5.4.

Comparing the total message counts and query success rates for R-DCS for different values of R and f, we see that R-DCS offers significant resilience to node failures with $R \geq 4$ for different values of f. In list mode, considering $N = 10000$, $E = 100$, $D_{total} = 10000$, $Q = 50$, $D_q = 5000$, $M = 16$, $f_c = 0.02$, $R = 4$ and $\omega = 0.1$, we get $T = 1029250, \quad Q_s = 1$ for $f = 0$ and $T = 1355032, \quad Q_s = 0.92$ for $f = 0.5$. With the other parameters constant but $R = 16$, we get $T = 839250, \quad Q_s = 1$ for $f = 0$ and $T = 1170410, \quad Q_s = 0.98$ for $f = 0.5$. Hence we see that as R is increased beyond 4, the query success rate Q_s approaches $1 - f_c$ as f is increased up to 0.5. The tradeoff here is the overhead of maintaining a large number of monitor/replica nodes.

Table 1. Simulation Parameters

Parameter	Value
N, Number of Nodes	100 - 100000
R, Number of Replicas	2-16
Z, Number of Zones	16
E, Number of Event Types	100
Q, Number of Event Types Queried	50
D_i, Instances of each Event-type i	100
f_c, Probability of Clustered Node Failure	0.02
ω, Frequency of Monitor Map Updates	0.1

5 Performance Evaluation

DCS has been shown to be a viable and robust data dissemination scheme using detailed simulations in $ns - 2$ extended with a GPSR-based DHT [7,9], which included a full 802.11 MAC and physical radio layer. These simulations verified the correct functioning of the low-level aspects of DCS. Since R-DCS builds upon DCS mechanisms, we expect that R-DCS will be viable in a bandwidth-limited, contention prone medium as well. However these $ns - 2$ simulations do not scale to more than 200 nodes [9]. Since we envisage R-DCS to be most useful in very large-scale sensor networks, we used a lightweight (*i.e.*, without radio details) simulator built in C to compare the performance of R-DCS with DCS, LS and ES. This simulator assumes stationary nodes in the sensor network, as well as instantaneous error-free packet transmission. We examine *total usage* and *hotspot usage*, as described before, as metrics to compare the relative performance of

these algorithms. The system parameters used in most of the simulations are summarized in Table 1.

We individually vary the parameters R, N, Q, and the node failure rate and investigate the effect on system performance. In our simulations, there is one randomly chosen access point, which represents the node where queries enter the network. At the start of the simulation, all events are inserted into the DHT once, by sensors chosen uniformly at random; these are the sensors that measured the inserted events. We present results averaged over multiple simulations (10 iterations with different random seeds for each simulation run).

5.1 Variation of R, the Number of Replicas

In this section, we show the results of varying the number of replicas R, while the other parameters are kept constant as in Table 1. The value of N is chosen to be 10000. Figure 5 shows the variation of total messages with R for the *list* and *summarized* cases. From Figure 5, we observe that ES and LS have higher total usages than our data-centric schemes. At low values of R, R-DCS and its extensions are found to have approximately similar performance as DCS. However, as R is increased, the performance of R-DCS based schemes is significantly better than DCS. These conclusions are also supported by our analytical results in Section 4. Note also that the total usage for R-DCS based schemes is significantly lower in *summarized* case than in the *list* case, as expected.

In our simulations, we first consider a *normal* scenario, where events occur uniformly spread out over the sensornet field. We have also conducted simulations with a *modified* event density, wherein 80% of the events occur in one quadrant of the field, and 20% of events occur in the remaining three quadrants taken together. We expect that this will lead to congestion in the "crowded" quadrant of the network. Our results (which we do not include here for lack of space) show that load-balancing mechanisms built into R-DCS help it outperform the other schemes in this case.

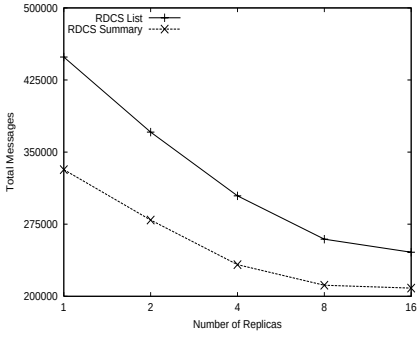

Fig. 5. Variation of Total Usage with R (ES=740206, LS=684409, DCS=488917)

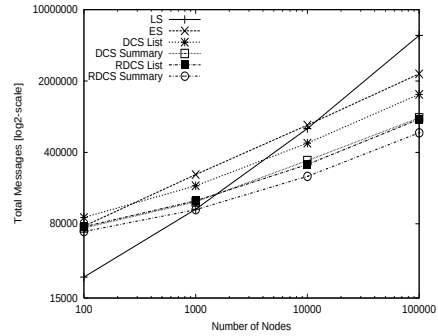

Fig. 6. Variation of Total Usage with Number of Nodes N

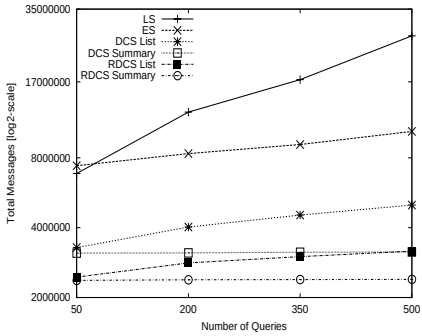

Fig. 7. Variation of Total Usage with Q

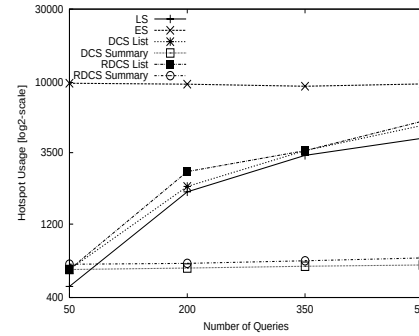

Fig. 8. Variation of Hotspot Usage with Q

5.2 Scaling to a Large Number of Nodes

In this section, we show the results of varying the number of nodes N, while the other parameters are kept constant as in Table 1. The value of R is chosen to be 4. Figure 6 shows the variation of total messages with n. From Figure 6, we see that all the methods have reasonably similar behavior for total usage, but LS has the lowest total usage at low N ($N = 100$) and ends up (at high N) with the highest value. Amongst the other schemes, our R-DCS schemes have the best performance, followed by DCS and then ES, for all values of N. These conclusions are supported by our analytical results in Section 4.

5.3 Scaling to a Large Number of Queries

In this section, we show the results of varying the number of queried event-types Q, while the other parameters are kept constant as in Table 1. The value of R is chosen to be 4 and $N = 10000$. Q is varied from 50 to 500. Since the number of instances of each event type is 100, this corresponds to a variation in the total number of queries from 5000 to 50000. Figure 7 shows the variation of total messages with Q, while Figure 8 shows the corresponding variation of hotspot usage. From Figures 7 and 8, we see that at low values of Q, LS offers good performance, but both total usage and hotspot usage increase linearly with increasing Q - hence LS does not scale well with more queries. ES has a medium total as well as hotspot usage, which is independent of Q. In both the *list* and *summarized* case, the graphs clearly show that R-DCS offers significant performance improvements over DCS which in turn does better than ES/LS with increase in Q. These experimental results are supported by our analytical expressions in Section 4, confirming the validity of our simulations.

5.4 Resilience to Node Failure

In the results presented so far, we assumed that nodes were stable. We now present simulation results for the case when nodes are *off* with probability f

and *on* with probability $1 - f$. We vary the value of f from 0 to 0.5 and observe the variation of T. The value of N is chosen to be 10000, and the other parameters are the same as in Table 1. Figure 9 shows the variation of total messages with f for the *summary* case in R-DCS. In fact for $R = 16$, T is almost constant for values of f ranging from 0 to 0.5. It must be noted that, as the node failure rate f is increased, the Query Success Rate Q_s is expected to decrease significantly as shown by Equation 9. For example, with $f = 0.4$ and $R = 2$, $Q_s = 70\%$ versus $Q_s = 100\%$ for $f = 0$. Our experimental results approximately agree with the analytical results derived in Section 4.2. These results shows that in a scenario where the probability of node failure is significantly high, R-DCS improves scalability by avoiding a dramatic increase in messages sent. In other words, R-DCS makes sensor data *resilient* to node failure.

6 Conclusions

In this paper, we have presented Resilient Data-Centric Storage as a means of reducing energy consumption and increasing resilience to node failures in a wireless ad-hoc sensor network. We present a methodology which stores event data at the closest of R replica nodes allocated (by the use of a DHT) for the event-type to which this data belongs. A set of monitor nodes exchange monitor maps to form a global image of all events of that event type which have occurred in the network. Since queries need to be routed to the closest monitor or replica node for an event-type, overall query traffic is reduced.

We have evaluated the viability and relative performance of our scheme vis-a-vis the original DCS scheme, Local storage and External storage. Through analytical results and simulations, we show that in a reasonably large sensor network with many detected events, R-DCS performs better than DCS, LS and ES. In particular R-DCS achieves significant energy-savings in the *summarized* mode of query responses. In all scenarios, R-DCS provides resilience to both clustered and isolated node failures.

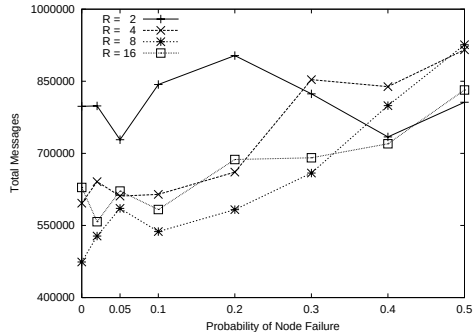

Fig. 9. Variation of Total Usage with node failure rate f

We believe that R-DCS will work well over a broad range of network scenarios. Robust and resilient storage of data within a sensor network could be useful for a large number of applications, such as monitoring meteorological data in turbulent conditions. Resilient DCS could also be used as a tool in providing service guarantees for applications running over sensor networks, analogous to how web-caching helps in providing service guarantees and improving data-availability over the Internet.

7 Acknowledgements

We would like to thank Sylvia Ratnasamy, Yin Li and Fang Yu for their help and guidance in the initial phases of the project and for making the source code of their DCS simulator available to us. We also thank the anonymous reviewers for their useful comments and feedback. This work is supported by the US National Science Foundation under Cooperative Agreement Number ITR-0085879.

References

1. W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The Design and Implementation of an Intentional Naming System. In *Proceedings of ACM SOSP-99*, pages 186–201, Dec. 1999.
2. B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, July 1970.
3. L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. In *Proceedings of ACM SIGCOMM'98*, pages 254–265, Sept. 1998.
4. J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *Proceedings of ACM SOSP-01*, 35(5):146–159, Oct. 2001.
5. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of ACM MOBICOM-00*, pages 56–67, Aug. 2000.
6. B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of ACM MOBICOM-00*, pages 243–254, Aug. 2000.
7. S. Ratnasamy, B. Karp, Y. Li, F. Yu, R. Govindan, S. Shenker and D. Estrin. GHT: A Geographic Hash Table for Data-Centric Storage. In *Proceedings of ACM WSNA-02*, Oct. 2002.
8. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM 2001*, 31(4):161–172, Aug. 2001.
9. S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-Centric Storage in Sensornets. In *Proceedings of ACM HotNets-I*, Oct. 2002.
10. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM 2001*, 31(4):149–160, Aug. 2001.