Resilient Data-Centric Storage in Wireless Sensor Networks

Abhishek Ghose **Jens Grossklags** **John Chuang**
University of California, Berkeley

Resilient Data-Centric Storage replicates data at strategic locations in wireless ad hoc sensor networks, leading to significant energy savings, as well as enhanced scalability as node density and query rates increase. R-DCS's performance also gracefully degrades when faced with isolated and clustered node failures, increasing data robustness.

IEEE Distributed Systems Online
Vol. 4, No. 11, November 2003

Wireless sensor networks make possible a range of applications that link numerous nodes to monitor and report distributed event occurrences. Examples include fields as diverse as climatic monitoring, tactical surveillance, and earthquake detection. When built with state-of-the-art technology, wireless sensors offer processing and storage capabilities that are significantly higher than the available bandwidth.[1] The reason is limited battery life, which generally motivates developers to conserve energy by using low-power—and, consequently, low-bandwidth—wireless communication techniques. Comparing these high processing capabilities with high communication costs encourages system designers to leverage computational techniques that reduce the network's total communication overhead.

Recently, researchers have suggested data-centric models as a viable approach to reduce communication traffic in sensornets. For example, in data centric-storage (DCS)[2] data is stored according to its event type at designated sensornet nodes. As a direct consequence, users can send queries for a particular data type directly to the node that stores that data type rather than flooding the network with queries. Our scheme, Resilient Data-Centric Storage (R-DCS), extends this idea by replicating data and control information within the sensornet. By increasing the number of nodes that can store data for each event type, as well as maintaining summary and control information at several nodes, we further decrease the average cost of both storing and querying data. Our scheme also provides significantly higher data availability for unreliable or randomly failing networks.

Data-access and attribute-based naming: Existing approaches

In most communication networks, naming of nodes enables low-level communication and leverages topological information. With the Internet’s point-to-point communication, for example, IP addresses assigned to each node serve as unique node identifiers in IP routing. In a sensor network scenario, such naming is ineffective because an individual sensor node’s identity is less important than its associated data. Using an attribute-based naming system more efficiently satisfies wireless sensor networks’ tight energy constraints.[1,3]

How these methods work

At the lowest level, when an event occurs, sensors record and store the event data locally and name this data based on its attributes—such as event type or geographic location. In an environmental sensing network, for example, we might classify all sensor data as being of type "temperature," "pressure," or "humidity" and name all such data by including in them one of these event attributes. The owner of a sensornet can either predefine these attributes to save overhead or update them if a new set of tasks or data become central to observation.

Data-access methods handle named data in different ways, with substantially different assumptions and cost-benefit trade-offs. In the three canonical approaches,[2] for example,

External storage stores event data at an external storage point.
Local storage stores event data locally, at the detecting node.
DCS stores event data at designated nodes, according to event type.

To retrieve the sensornet’s event information, all methods except external storage use queries. It’s thus important to consider the ratio of query traffic to event detection traffic when designing a sensor network. R-DCS and each of the canonical approaches have different relative costs for query and event traffic. Hence, a sensor network’s function typically determines which data-access method offers the most advantageous energy management combined with adequate data availability.

Data-centric storage

We built our R-DCS schema on the DCS data-dissemination paradigm and its supporting mechanisms.[2] DCS provides access to data using an attribute-based naming scheme that builds a distributed hash table (DHT)[4-7] on top of a variation of the Greedy perimeter stateless routing (GPSR) algorithm[8] for low-level geographic routing.

Geographic routing—unlike the commonly used shortest path technique—uses the relationship between location and connectivity in a wireless network. GPSR uses greedy forwarding to forward packets to nodes that are progressively closer to the destination. In regions where such a greedy path does not exist—that is, where the only viable path requires a move away from the destination—GPSR recovers by forwarding in perimeter mode, wherein a packet traverses successively closer planar subgraph faces within the network connectivity graph until it reaches a new node closer to the destination, and greedy forwarding resumes.

On top of GPSR, DCS uses a DHT to hash an event name’s key (dataName) to geographic coordinates located within the sensornet’s boundaries. DCS offers two interface primitives:

- *put(dataName, dataValue)* stores dataValue (the observations of a sensor node with a particular attribute) under the key dataName.
- *get(dataName)* retrieves all data stored in the sensornet that is associated with the key dataName.

Using GPSR, DCS routes a `put` or `get` message to the node closest to the key dataName's hashed location. This node is expected to act as the storage node for data associated with the key. Although this node need not be at the exact coordinates that the hash function determines, it is important that the node's identity is chosen in a consistent manner.

Thus, to make DCS capable of handling limited mobility and isolated node failures, the DCS developers have proposed certain extensions to its basic scheme.[2,9] In DCS, an event type's storage node periodically initiates a GPSR-routed refresh message to the hashed location of its own key dataName that contains all associated data at this node. If the refresh message detects a new or displaced node closer to the key dataName's associated location than the current storage node, this new node becomes the new storage node. Also, timer-based algorithms ensure that f an active storage node dies unexpectedly, another node automatically starts generating refresh messages. Thus, to protect against isolated node failures, the nodes that receive a local refresh message also must replicate the contained (dataName, dataValue) pairs to protect against data loss. This process is called the *perimeter refresh protocol* (PRP).[9]

Resilient data-centric storage

R-DCS is an extension to DCS's basic functionality that further minimizes query retrieval traffic and thus conserves energy and increases data availability. R-DCS ensures that event information is not lost, even if multiple nodes or whole regions fail during operation. For example, because the original version of DCS stores all events of the same event type in one sensornet node, too many events of a particular type can create a network bottleneck, or *hotspot*. Also, clustered node failures can lead to a complete data loss for a certain event type. Our R-DCS data-dissemination scheme contends with this using a two-level replication strategy for control information and data. R-DCS facilitates data storage at one of several *replica* nodes in the network assigned to an event type. It also facilitates storage of this event type's control and summary information at geographically distributed *monitor* nodes in the network.

R-DCS architecture

In R-DCS, we partition the sensornet field's coordinate space into Z zones, denoting the set of available zones as z_j: $j = 1, \dots, Z$. Figure 1a shows an example of zoning based on geographical boundaries. These zones can contain sensor nodes operating in three possible modes: monitor, replica, and normal.

Figure 1. The R-DCS architecture. The sensornet field is partitioned into zones (z) that include monitor (m) and replica (r) nodes, and might also contain an access point (AP). (a) R-DCS event storage, (b) querying in list mode, (c) querying in summary mode, and (d) a logical ring.

Each zone has one *monitor* node for each event type. The monitor node uses a monitoring map to store and exchange information for each event type. This map includes control and summary information in several fields:
A list of zones containing replica nodes (to forward event data and queries).
A list of zones containing monitor nodes (to facilitate map exchange).
Event summaries (to facilitate summary-mode queries). The exact nature of event summaries depends on the event type. For example, in a temperature-monitoring sensornet, the summary information might contain the number of events detected and the average temperature reading for each zone.
Bloom filters (to enable attribute-based queries; this field is optional). R-DCS uses Bloom filters[10,11] to organize event data as a set of attributes and their values. In the temperature monitoring case, for example, these attributes might be `Event time` and `Temperature`. Bloom filters offer an efficient way to support attribute-based queries. For example, they let users access all temperature measurements conducted between 3 p.m. and 4 p.m., or all temperature readings between 30 and 40 degrees.

Each zone also has at most one *replica* node for each event type. If present, this replica node is always the same as the monitor node, performing its functions but also storing event data for the given event type. All nodes that are not either a monitor or replica node operate in *normal mode*. Such a node might originate or route event data, but doesn't store any event data or control information (except as a part of PRP).

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. For each $i = 1, \ldots, E$, we must satisfy the following system constraints:

$R_i \leq M_i \leq Z$. This holds because each zone has 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 (when a majority of nodes fail in one zone), there will be one monitor node per zone: $M_i = Z$.
$R_i \geq 1$. The network must have at least one replica node, because a replica node stores each event type's event data.

Due to the higher complexity of R-DCS, for our DHT we use a hash function H that is a function of the event type e_i and also the zone z_j. If event type e_i in zone z_j hashes to a location (x_{ij}, y_{ij}) º $H(e_i, z_j)$, then the 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, this monitor node might also serve as a replica node r_{ij}. For load balancing, the function $H(e_i, z_j)$ should be chosen such that for each zone z_j, different event types e_i hash to distinct nodes.

Event Storage and Query and Monitoring Map Update

To store events, 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 event type's closest replica node, which is determined based on the information in the local monitoring map's *list of replica nodes*. The target replica node stores the event data and updates its local copy of the monitoring map (Figure 1a).

R-DCS makes possible three types of event queries: summary, list, or attribute-based. A *list query* requests all stored data 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} (which might be identical to a local replica node). The monitor node then duplicates the query and forwards it to all other active replica nodes $r_{ix} : x = 1, \ldots, Z$ in the sensornet. All active replica nodes reply directly to the querying node with event data (see Figure 1b).

A *summary query* 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 (see Figure 1c).

An *attribute-based query* requests data for all events that match certain attribute-value constraints. 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. All active replica nodes reply directly to the querying node with event data.

When an event of type e_i occurs in zone j, it updates the local monitoring map in m_{ij}. For global information consistency, all active monitor nodes for a type e_i form a logical ring to periodically exchange monitoring maps (see Figure 1d). 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 using knowledge about its two adjacent zones.

Switching between modes

A node in zone j switches from normal mode to monitor or replica mode (for event type i) when it becomes the node closest to the location (x_{ij}, y_{ij}) º $H(e_i, z_j)$. During this switch, the retiring monitor node hands over the monitoring map, and, in the case of a replica node, any relevant stored data.

A node's switch from monitor mode to replica mode and vice-versa are based on certain local criteria, such as event storage and query traffic loads. The switch is also related to the node's residual energy: sensornet nodes have limited energy, so when a replica node's energy is running low, it will try to conserve energy by becoming a monitor node.

Handling node failure

The above discussion assumes that all nodes are stable and able to continuously route, monitor, or store data. However, it's more realistic to consider node failure, with a certain failure probability f.

When a monitor node in a certain zone fails, we can use PRP to find and 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 a replica node, PRP can recover at least part of the node's stored event data through local refreshes.

Under normal operating conditions, there is one monitor node per zone—that is, $M_i = Z$ for each event type e_i. These nodes form a logical ring for map updating. However, if a *clustered node failure* occurs (all or most nodes in a particular zone fail or are disabled), the logical ring of monitor nodes breaks. In each map-update cycle, the monitor nodes mark their presence by updating the monitoring map's field listing monitoring nodes. By checking this field, it is possible to discover which zones have clustered node failures and route around them to reconstruct a logical ring of $M_i < Z$ monitor nodes. To recover from such failures, we can use different techniques, such as beaconing mechanisms.

Performance measures

To quantify R-DCS's benefits, we must specify relevant performance measures. Data dissemination algorithms ideally seek to minimize communication and thus extend the overall system lifetime. For the sake of consistency and easy comparison, we quantify communication overhead using the same metrics as Sylvia Ratnasamy and her colleagues.[2,9] Assuming same-sized packets, these metrics are

Total usage: the total number of packets sent in the sensornet
Hotspot usage: the maximal number of packets sent by any particular sensornet node

As we mentioned earlier, when a sensornet experiences a particular event type, it transmits the event data to the closest monitor node, which then forwards the data to the closest replica node for storage. If the data storage operation fails—if, for example, the destination node is unreachable or malfunctioning—the monitor node retransmits the data to another R_i replica node. This process iterates until the data is successfully stored. If all the R_i replica nodes for event type i fail simultaneously, then the attempt to store or retrieve data of this type will fail.

To measure the success rate of event data queries, we use two metrics:

Query success rate measures the mean percentage of queries (averaged over all event types) that return a successful response.
Total usage (as a function of node failure rate f) serves as a resilience metric.

Although we expect total usage to increase as the node failure rate increases (due to more retransmissions), 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. Thus, our aim is to achieve a graceful degradation in system performance as f increases.

Analytical results

We can now derive analytical expressions for the energy costs and savings obtainable with R-DCS. For this analysis, 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 the number of replica nodes, M the number of monitor nodes, and ? the frequency of monitoring map updates.

Energy savings

We can estimate R-DCS's communication costs in relation to the three canonical methods[10] using the asymptotic cost of a flood, $O(N)$, and that of direct GPSR routing from one random node to another:

$O(\sqrt{N})$.

For R-DCS, we estimate the total message count (total usage T) and the number of messages at the busiest point (hotspot usage H). For completeness, we reproduce the corresponding expressions—and their expected values—for each of the aforementioned canonical methods from existing work.[10] In our scenario, there is one access point, but results can be easily generalized for multiple access points. We assume that the access point's message count is a good estimate of hotspot usage, since it's likely to be the network's busiest area.

External storage

The cost of sending each event to external storage is $O(\sqrt{N})$. There is no cost for queries, because event information is already external.

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

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 $O(\sqrt{N})$.

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

Data-centric storage

Depending on the event type, storing event information at a sensornet node incurs a cost of $O(\sqrt{N})$. The access point sends queries for a particular event type to the storage node, which returns a response; the cost of both is $O(\sqrt{N})$. Total usage depends on whether a full event listing is required (using one packet for each instance of each event type) or whether a summary is sufficient (using only one packet for each event type). For list and summary, respectively

$$\begin{aligned} T &= D_{total}\sqrt{N} + Q\sqrt{N} + D_q\sqrt{N} \qquad H = Q + D_q \\ T &= D_{total}\sqrt{N} + Q\sqrt{N} + Q\sqrt{N} \qquad H = 2Q \end{aligned} \tag{3}$$

For completeness, we also report the analytical results for structured replication with DCS (SR-DCS), which is another DCS extension that achieves load balancing in the network by creating a hierarchical decomposition of the key space and associates each event type with a hierarchy depth. Storage is done at the hierarchy's closest node, whereas queries still have to be routed through the hierarchy's entire node set. Thus, it is obvious that SR-DCS decreases storage costs and increases query costs. Also, SR-DCS does not involve actual data replication, but it increases DCS's scalability. 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 the original DCS to $O(2^d\sqrt{N})$ for SR-DCS. The following equations give total usage for list and summary, respectively:

$$T = D_{total}\sqrt{N}/2^d + Q2^d\sqrt{N} + D_q2^d\sqrt{N} \qquad H = Q + D_q$$
$$T = D_{total}\sqrt{N}/2^d + Q2^d\sqrt{N} + D_q2^d\sqrt{N} \qquad H = 2Q \qquad (4)$$

Resilient data-centric storage
Because R-DCS stores event information at the node closest to the event location, storage costs are reduced from $O(\sqrt{N})$ to $O(\sqrt{N/R})$. The access point sends queries of a particular event type to the closest monitor node at a cost of $O(\sqrt{N/M})$. In the summary case above, this monitor node can directly return a response (based on monitor maps that other nodes send it) at a cost of $O(\sqrt{N/M})$. In the list case, the monitor node must forward the query to all *R* replica nodes, at a cost of $O(R\sqrt{N})$, and all replica nodes must return responses, at a cost of $O(\sqrt{N})$. Let the frequency of monitoring map updates be ? .

The cost of exchanging monitor maps is $O(2\omega\sqrt{NM})$ for each event type.

$$T = D_{total}\sqrt{N/R} + Q(\sqrt{N/M} + R\sqrt{N}) + D_q\sqrt{N} + 2\omega E\sqrt{NM} \qquad H = Q + D_q$$
$$T = D_{total}\sqrt{N/R} + Q\sqrt{N/M} + Q\sqrt{N/M} + 2\omega E\sqrt{NM} \qquad H = 2Q \qquad (5)$$

From these formulas, we can draw several conclusions. If *N* is increased and other parameters are constant, the local storage method incurs the highest total message count. If D_q » Q and events are summarized, DCS and R-DCS have the lowest hotspot usage. If D_{total}» D_q, external storage has significantly higher hotspot usage than the other schemes. In general, DCS and R-DCS are preferable when *N* is large or D_{total} » D_q » Q (that is, there are many detected events and not all event types are queried). However, local storage might be preferable if there are many events relative to system size, $\left(D_{total} > Q\sqrt{N}\right)$, and event lists are used.

Comparing the total message counts for R-DCS, DCS, and SR-DCS (the *R* in R-DCS is equivalent to 4^d in SR-DCS; that is, we compare the number of R-DCS replicas with the total number of nodes in an SR-DCS scenario's hierarchy) we see that R-DCS could provide significant energy savings over DCS for both the summarized and list cases. R-DCS presents an intermediate solution between local storage (free storage, expensive queries) and DCS (both at moderate cost). When D_q » Q, we expect R-DCS to offer good performance.

Increased resilience

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

Let's consider a situation in which sensor network nodes are unreliable; they are on (functioning correctly) with probability $1 - f$, and off (malfunctioning) with probability f. Here, $0 \leq f \leq 1$, and $f = 0$ indicates perfectly reliable nodes. As in the previous section, we approximate R-DCS's total communication costs (T). We use local refresh mechanisms such as PRP to ensure that each zone has one monitor node, except in zones with clustered node failures. *Normal* nodes in any zone j communicate with the local monitor nodes m_{ij} (or local replica nodes, r_{ij}, if they exist) for data of event type e_i. In any message, this 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 that the local monitoring map lists; it then orders the R_i replica nodes according to their distances from the source. Hence, it first chooses the closest replica node as the destination, followed by the second-closest replica node, and so on. m_{ij} then routes the message to each of the possible R_i destinations in its list, in order of preference. If all R_i destinations are exhausted without success, the message is dropped.

With R-DCS, 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 it sends the message to the first alternate destination. In the second transmission, with probability $1 - f$, it successfully sends the message at a total cost of $2O(\sqrt{N})$—that is, 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. Equation 6 represents this iterative process.

GET EQUATION 6

where C_{Rf} is given by

GET EQUATION 7

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

GET EQUATION 8

P_s denotes the probability of successfully storing a particular event instance's data (here, we omit the subscript i, but we're considering a particular event type e_i). We denote the probability of clustered node failure as f_c. As we described earlier, this operation succeeds if all the event type's R replica nodes 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 must be satisfied:

(1) The queried event data must have been successfully stored earlier (which occurs with probability P_s).
(2) The local monitor node must be alive (storing this summary data in real time).

Because we assume that local refresh mechanisms like PRP ensure monitor node existence (except in clustered node failures), this monitor node is alive with probability $1 - f_c$. So, the mean success rate Q_s for summary queries is given by

$Q_s = (1 - f^R)(1 - f_c)$. (9)

For a list query to be successful, the second condition is modified to specify that all monitor nodes must be alive. This occurs with probability $(1 - f_c)^M$. So, the mean success rate for list queries is given by

$\mathbf{Q_s = (1 - f^R)(1 - f_c)^M}$ **.** **(9)**

From Equations 7 and 8, we can see that if f increases with other parameters constant, then T increases. This should be obvious intuitively, because a greater number of unreliable nodes imply more retransmissions. Also, Equations 9 and 10 imply that if R increases (keeping f constant), Q_s increases. Similarly, as we discuss in more detail in the next section, if f increases, Q_s decreases (for the same value of R). 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.

Performance evaluation

DCS proved to be a viable and robust data dissemination scheme on detailed simulations in the $ns - 2$ network simulator extended with a GPSR-based DHT,[2,9] which included a full 802.11 medium access control layer and physical radio layer. These simulations verified that DCS's low-level aspects functioned correctly. Because R-DCS builds on DCS mechanisms, it should be viable in a bandwidth-limited, contention-prone medium as well. However, these $ns - 2$ simulations don't scale to more than 200 nodes,[2] and we envision R-DCS as most useful in large-scale sensor networks. We thus built a lightweight simulator (without radio details) in C to compare R-DCS's performance with that of DCS, local storage, and external storage. This simulator assumes that the sensor network has stationary nodes and instantaneous, error-free packet transmission. To compare the algorithms' performance, we used the *total usage* and *hotspot usage* metrics. Table 1 summarizes the system parameters for most of the simulations.

Table 1. System parameters for simulations.

Parameter	**Value**
N (number of nodes)	100–100,000
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
? (frequency of monitor map updates)	0.1

We individually varied the parameters R, N, Q, and the node failure rate and investigated the effect on system performance. Our simulations had one randomly chosen access point that represented the node where queries entered the network. At the simulation's start, randomly chosen sensors inserted all events into the DHT once; these sensors measured the inserted events. We averaged our results over multiple simulations (10 iterations, with different random seeds for each simulation run). Unless we note otherwise, the node density remained constant (we scaled the region size as the number of nodes increased).

Variation in replica numbers

In one experiment, we varied the number of replicas *R* while keeping the other parameters constant (as in Table 1). We set the value of *N* at 10,000. Figure 2a shows the variation of total messages with *R* for the list and summarized cases. As the figure shows, external and local storage had higher total usages than our data-centric schemes. At low values of *R*, R-DCS and its extensions performed similar to DCS. However, as *R* increased, R-DCS-based schemes performed significantly better than DCS. Our analytical results support these conclusions. As expected, the total usage for R-DCS-based schemes is significantly lower in the summarized case than in the list case.

Figure 2. Variations in total messages. When we (a) varied the number of replicas, external storage had a total usage of 740,206, local storage's total usage was 684,409, and DCS's total usage was 488,917. Results are also shown for a varying number of (b) nodes, (c) queried event types, and (d) node failure rates.

In our simulations, we first considered a normal scenario in which events occurred uniformly across the sensornet field. In later simulations where we modified event density, 80 percent of the events occurred in one field quadrant, while the remaining 20 percent occurred across the remaining three quadrants. This will likely create congestion in the crowded network quadrant. The load-balancing mechanisms in R-DCS helped it outperform the other schemes in this case. In Figure 2a, we present the results for *T* averaged across scenarios with and without varying density.

Scaling to many nodes

In another experiment, we varied the number of nodes *N*, while keeping the other parameters constant. We set the value of *R* at 4. Figure 2b shows the variation of total messages with *N*. All methods had reasonably similar behavior for total usage, but local storage had the lowest total usage at low *N* ($N = 100$) and ended up (at high *N*) with the highest value. Among the other schemes for all values of *N*, R-DCS schemes performed best, followed by DCS, and then external storage. Again, our analytical results support these conclusions.

Scaling to many queries

In another experiment, we varied the number of queried event types *Q* while keeping the other parameters constant. We set the value of *R* at 4 and *N* at 10,000, and varied *Q* from 50 to 500. Each event type had 100 instances, which corresponded to a variation in the total number of queries from 5,000 to 50,000. Figure 2c shows the variation of total messages with *Q* (we omit the graph for hotspot usage, but summarize the results). Local storage offered good performance at low values of *Q*, but both total usage and hotspot usage increased linearly with increasing *Q*—so local storage doesn't scale well as queries increase. External storage had a medium total and hotspot usage, independent of *Q*. In both the list and summarized case, R-DCS clearly offered significant performance improvements over DCS, which in turn did better than external and local storage as *Q* increased. Our analytical expressions support these experimental results, confirming the validity of our simulations.

Resilience to node failure

Thus far, our results assumed that nodes were stable. We also ran a simulation in which the nodes were off with probability *f* and on with probability $1 - f$. We varied the value of *f* from 0 to 0.5 and observed *T*'s variation in a scenario without PRP. We set the value of *N* at 10,000, keeping the other parameters constant. Figure 2d shows the variation of total messages with *f* for the summary case in R-DCS.

For $R = 16$, *T* is almost constant for values of *f* ranging from 0 to 0.5. As the node failure rate *f* increases, we expect the query success rate Q_s to decrease significantly, as Equation 9 shows. For example, with $f = 0.4$ and $R = 2$, $Q_s = 70$ percent; when $f = 0$, $Q_s = 100$ percent. These experimental results are similar to our analytical results, which show that in scenarios with a significantly high probability of node failure, R-DCS improves scalability by avoiding a dramatic increase in messages sent. In other words, R-DCS makes sensor data resilient to node failure.

Conclusion

As our results show, R-DCS reduces energy consumption and increases node-failure resilience in wireless ad hoc sensor networks, and should perform well in a broad range of network scenarios. Such robust and resilient data storage in sensor networks could be useful in numerous applications, such as monitoring meteorological data in turbulent conditions. R-DCS could also act as a tool to guarantee service for applications running over sensor networks, just as Web caching helps provide service guarantees and improve data availability over the Internet.

Acknowledgments

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

References

1. J. Heidemann et al., "Building Efficient Wireless Sensor Networks with Low-Level Naming," *Proc. 18th ACM Symp. Operating Systems Principles* (SOSP-01), ACM Press, 2001, pp. 146–159.
2. S. Shenker et al., "Data-Centric Storage in Sensornets," *Proc. 1st ACM SIGCOMM Workshop on Hot Topics in Networks*, ACM SIGCOMM *Computer Comm. Review*, vol. 33, no. 1, Jan. 2003, pp. 137–142.
3. W. Adjie-Winoto et al., "The Design and Implementation of an Intentional Naming System," *Proc. 17th ACM Symp. Operating Systems Principles* (SOSP 99), ACM Press, 1999, pp. 186–201.
4. S. Ratnasamy et al., "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM 2001 Conf.*, ACM Press, . 2001, pp. 161–172.
5. A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-Peer Systems," *Middleware 2001*, LNCS 2218, Springer-Verlag, 2001, pp. 329–340.
6. I. Stoica et al., "Chord: A Scalable Peer-To-Peer-Lookup Service for Internet-Applications," *Proc. ACM SIGCOMM 2001 Conf.*, vol. 31, no. 4, ACM Press, 2001, pp. 149–160.
7. B. Zhao, J. Kubiatowicz, and A. Joseph, *Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing*, tech. report UCB/CSD-01-1141, Computer Science Dept., Univ. California, Berkeley, Apr. 2001.
8. B. Karp and H.T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. 6th Ann. Int'l Conf. Mobile Computing and Networking* (MOBICOM-00), ACM Press, 2000, pp. 243–254.
9. S. Ratnasamy et al., "Data-Centric Storage in Sensornets with GHT, A Geographic Hash Table," *Mobile Networks and Applications*, vol. 8, no. 4, Aug. 2003, pp. 427–442.
10. B.H. Bloom. "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM*, vol. 13, no. 7, July 1970, pp. 422–426.
11. L. Fan et al., "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *Proc. ACM SIGCOMM Conf.*, ACM Press, 1998, pp. 254–256.

Abhishek Ghose performed this research as a graduate student in the Department of Electrical Engineering and Computer Sciences at the University of California at Berkeley. His research interests include wireless sensor networks and performance measurement in computer networks. He has an MS in electrical engineering and computer sciences from the University of California at Berkeley. Contact him at abhishek_ghose@yahoo.com

Jens Grossklags is a PhD student in the School of Information Management and Systems at the University of California at Berkeley. His research interests include computer networks, economics, and human factors research. He has a diploma in business administration with an emphasis in economics and information systems from Humboldt-University Berlin. Contact him at jensg@sims.berkeley.edu.

John Chuang is an assistant professor in the School of Information Management and Systems at the University of California at Berkeley. His research interests encompass the technical and economic dimensions of computer networking. He has a PhD in engineering and public policy from Carnegie Mellon University. Contact him at chuang@sims.berkeley.edu.