Data-Centric Storage in Sensornets

Sylvia Ratnasamy, ICIR
Deborah Estrin, UCLA
Ramesh Govindan, ICIR
Brad Karp, ICIR
Scott Shenker, ICIR
Li Yin, UC Berkeley
Fang Yu, UC Berkeley

Abstract

Sensornets are large-scale distributed sensing networks comprised of many small sensing devices equipped with memory, processors, and short-range wireless communication. Making effective use of sensornet data will require scalable, self-organizing, and energy-efficient data dissemination algorithms. Recent work has identified data-centric routing as one such method. In this paper we suggest that a companion method, data-centric storage, may also be a useful approach.

While there are many ways to achieve data-centric storage, this paper proposes a mechanism that builds upon two recent advances; (1) the GPSR geographic routing algorithm and (2) a new generation of efficient peer-to-peer lookup systems (such as Chord, CAN, Pastry, Tapestry, etc.). We evaluate the performance of data-centric storage and two other dissemination approaches in several sensornet scenarios and identify the conditions under which the various approaches are preferable.

1 Introduction

Recent engineering advances in the design of small energy-efficient hardware and compact operating systems have enabled the development of large-scale distributed sensing networks (sensornets) made up of many small sensing devices equipped with memory, processors, and short-range wireless communication. These sensornets will provide an unprecedented amount of detailed measurements over wide geographic areas; however, this data is distributed across the entire sensornet, making it hard to access. Communication between sensornet nodes requires the expenditure of energy, a scarce commodity for most sensornets. Thus, making effective use of sensornet data will require scalable, self-organizing, and energy-efficient data dissemination algorithms.

Since the nature of the data is more important than the identity of the node that gathers it, sensornet researchers have found it useful to move away from the Internet's point-to-point communication abstraction and instead adopt abstractions that are more data-centric. This approach entails "naming" the data and then using communication abstractions that refer to those names rather than to node network addresses [12, 1]. In particular, previous work on data-centric routing has shown it to be an energy-efficient data dissemination method for sensornets [14]. Here, we suggest that a companion method, data-centric storage, may also be useful. In this approach, relevant data is stored, by name, at nodes within the sensornet; all data with the same general name (*e.g.,* elephant sightings) will be stored at the same sensornet node (not necessarily the one that originally gathered the data). Queries for data with a particular name can then be sent directly to the node storing that named data, thereby avoiding the flooding required in some data-centric routing proposals.

There are several data-centric dissemination methods, and these offer rather different performance characteristics. Which one is appropriate for a particular setting will depend on the nature of the sensornet and its use. Consequently, our claim here is not that data-centric storage is always the method of choice, but rather that under some conditions it will be preferable. In fact, we expect that future sensornets will embody all of these (or similar) data-centric dissemination methods, and that users will choose which to use at any particular time based on the task at hand.

This paper has two main foci. Our first focus is on identifying under what circumstances data-centric storage is the preferred method. This requires that we begin our paper with a general but lengthy discussion of sensornet dissemination algorithms and the settings in which they might be used. This will provide the needed context for the later comparative simulations.

Our second focus is on providing a mechanism for achieving data-centric storage. Our proposed mechanism is based on two recent developments. First, it uses the GPSR geographic routing algorithm [16] as a low-level routing mechanism. Second, at a higher level it uses the hash-table like functionality provided by the new generation of peer-to-peer lookup algorithms such as Chord, CAN, Pastry, and Tapestry [27, 24, 8, 32]. This proposed method is a proof-of-concept rather than a finely tuned mechanism; several aspects of our design could conceivably be improved upon. In addition, as we discuss later, one can also use other techniques to achieve data-centric storage; in particular, the GRID geographic location system (GLS) [19] algorithm can be modified to provide data-centric storage. The SCOUT system proposed a similar approach [18].

Our paper has 6 sections. We start with a general discussion of sensornets, including related work, in Section 2. We then, in Section 3, describe a sensornet scenario in which data-centric storage is appropriate. We detail our mechanism for data-centric storage in Section 4 and evaluate it in Section 5 through two sets of simulations, one at the ns-level to confirm the correctness of

the low-level details and the other at a higher less-detailed level to compare it to two other dissemination approaches. Section 6 concludes the paper with a short discussion.

2 Context and Related Work

In this section we first state our basic assumptions about the class of sensornets we consider. We then describe some basic concepts used in organizing sensornet data. This is followed by a discussion of related work, which ends with a description of some possible approaches to data dissemination in sensornets.

2.1 Assumptions

While sensor node technology is advancing rapidly, and experiments have been run on small-scale sensornets [7], no large-scale sensornets have yet been deployed in practice. Thus, most papers on sensornet algorithms are hypothetical, in that they discuss scenarios for sensornet deployment (and sometimes node capabilities) that have not yet been deployed in the field (although they may have been deployed in testbed environments). This paper is no different; when we speak about sensornets, we are referring to the projected and experimental designs and deployments discussed in the literature and not to actual instances of sensornets deployed in the field.

These projected sensornet designs differ greatly in their characteristics and intended use. Here we focus attention on a certain class of sensornets that is most relevant to the issues we address. There are many other sensornet designs and uses that fall outside of our purview; thus, our discussion of assumptions, both immediately below and later in this paper, should not be interpreted as general statements about all sensornets but rather only as statements defining the class of sensornets we are discussing.[1]

We consider sensornets where the nodes are spread out, randomly or in some pattern, over some well-defined area. We assume that the sensornet operators know rough approximations to the geographic boundaries of the sensornet and that the area within those boundaries is fairly densely covered with nodes.

We assume that nodes only have short range communication, but are within radio range of several several other nodes. We further assume that the nodes know their location (in what follows, this can be either their relative or their absolute location). This can be achieved through the use of GPS or some other approximate but less burdensome localization algorithm; see, for example, [4, 11, 22, 23, 25]. This assumption that nodes know their (absolute or relative) location is crucial for our proposed data-centric storage mechanism. However, we think it is a reasonable assumption because in many cases the sensornet data is useful only if the location of its source is known; *e.g.*, knowing that a certain temperature reading occurred may not be useful if you don't know *where* it occurred.

We assume that energy is a scarce commodity for sensornet nodes [21]. Using a node's wireless communication requires energy and so the data dissemination algorithms should seek to minimize communication in order to extend overall system lifetime. While the mapping between communication and energy consumption is complicated – depending greatly on the precise hardware involved and the packet transmission pattern – in what follows we will focus on two simplified metrics of energy conservation:

Total usage: The total number of packets sent in the sensornet

Hotspot usage: The maximal number of packets sent by any particular sensornet node

We further assume that communication to the outside world takes place through one or more access points in the sensornet, and so getting data from a sensornet node to the outside world requires sending the data through the sensornet to the access point. We use the term *access path* to refer to the set of paths data from sensor nodes to the access point(s).

We further assume that nodes are reasonably stationary and stable. That is, while node failures do occur, and nodes might occasionally move (*e.g.*, blown by the wind, etc.), such occurrences are not the common case and happen on time scales long compared to packet transmission times. Similarly, we assume that the radio channel characteristics do not vary rapidly in time; our proposed mechanism makes no particular assumption about radio propagation other than that the set of nodes within radio range of a particular node doesn't change very rapidly. For settings where the set of reachable nodes does change rapidly, other approaches will be needed.

While in the rest of the paper we treat all nodes as having the same capabilities, it is likely that real sensornets will have a tiered architecture, with some nodes having very limited data storage capacity and others having much more significant storage (and perhaps also more battery power and better communication facilities). Our discussion applies to this tiered approach as well, as long as the these "macronodes" are numerous and widely dispersed [5].

The above assumptions described the physical nature of the sensornet. We next discuss how these sensornets might be used.

2.2 Observations and Events

The purpose of sensornets is to provide detailed sensing capabilities across a wide geographic area. The low-level output from these sensors, which we call *observations*, is named (as described, for example, in [12, 1]). The overwhelming volume of observations produced by these sensors is both a blessing and a curse. While sensornets give unprecedented access to the detailed observations of the physical world, sending these observations directly to the access point(s) would quickly exhaust the energy reserves of the sensornet. However, in most cases users don't want the complete set of raw unprocessed data, but rather are more interested in specific *events*, such as earthquakes, animal sightings, or bomb detonations.

We use the term *events* to refer to certain constellations of low-level observations. The constituent observations may span time,

[1] In other words, when we say we assume something, we mean we only consider cases where that statement holds.

space, and modalities; *i.e.,* the observations may be recorded at different times, at different places, and from different types of sensors (*e.g.,* temperature, humidity, air pressure, etc.). A sensornet system will be designed to detect several well-defined *types* of events.

Detecting these events requires some processing of the low-level observations. Sending all the data to the access point and processing the low-level observations externally would be very expensive in terms of energy consumption, both in terms of the total usage and in terms of the hotspot usage along the access path. To avoid this, sensornets can use *collaborative information processing* within the sensornet to produce high-level events from low-level observations. This may involve a combination of purely local techniques (*e.g.,* exchanging data with nearby nodes) and nonlocal techniques (*e.g.,* comparing temperature readings against the average readings in the sensornet). We will discuss these techniques in a bit more detail in Section 2.4.

Events can be defined not only in terms of low-level observations but also in terms of other events. For instance, detecting an animal migration would involve many individual animal sightings. In general, there will be a *web* of events, with some events defined in terms of others. These events are not necessarily in a strict hierarchy, but in the context of a particular application there is some sense that some events are lower-level than others, and could be used to define the higher-level events.

Once an event has been detected, users might want to access the low-level observations related to that event. For instance, if a volcanic eruption has been detected, the detailed temperature, humidity, and air quality readings of the various sensors in the area would be of great scientific value. Each event notification includes the event's location, so that to gather additional data one need only download the observations from the relevant sensors. Thus, event notification is not only useful in itself but is also a way to efficiently direct subsequent queries to the relevant data, thereby keeping the overall communication load manageable.

2.3 Tasks, Actions, and Queries

The preceding discussion identified the various types of information – observations and events – that might be provided by sensornets. We now describe how this information might be extracted from the sensornets.

Users send instructions (by flooding or some other global dissemination method) to sensornet nodes to run certain local identification *tasks*. These tasks could be simple, such as taking temperature readings, or complex, such as identifying an animal from a collection of sensor readings. In essence, one can think of tasks as downloaded code.

Once an event has been identified, nodes can take a number of different *actions*. For example, a node could be instructed to send event information to external storage, or to store the event information locally, or to use data-centric storage. Recall that data-centric storage involves storing the event information at a sensornet node that is chosen based on the event's name. These three possible actions – external store, local store, and data-centric store – will form the core of the three canonical approaches we describe in Section 2.5 and compare in Section 5.

Unless the information has been sent to external storage, at this stage the event information is still not in the user's hands. *Queries* are used to elicit the event information from the sensornet. How queries are executed will depend on the actions nodes take upon event detection. If event information is stored locally then queries must be flooded to all nodes (unless the user has prior knowledge about the location of the event). If event information is stored using data-centric storage, the query can be sent to the sensornet node associated with that event name. Note that queries can be either externally generated by users or internally generated by sensornet nodes engaged in event detection. As an example of the latter, a node may generate a query for recent sightings of an animal before making an identification.

2.4 Related Work

Tasks, actions, and queries are useful concepts, but we have yet to describe the actual protocols and algorithms used by sensornets. We now discuss those algorithms by briefly reviewing some of the related work on sensornets. We organize this discussion in "layers" ordered from bottom to top. However, these layers are used only to clarify the presentation; we don't mean to imply that sensornet architecture is organized into clean well-separated layers. Given the extensive scope of related work, the listing of related work below is illustrative but by no means exhaustive.

L1: Physical and OS: There have been numerous engineering advances in producing small and energy-efficient nodes with non-trivial amounts of memory, processor power, and sensors [21, 13]. In addition, novel operating systems have been designed specifically for such nodes [13].

L2: Low-level communication and self-configuration: Underlying all short-range communication is a MAC layer. The need for energy efficiency in sensornets may require new media access protocols that coordinate node duty cycles in an effort to save energy [30], or that regulate access to the medium in a fair way when a majority of the nodes simultaneously transmit their readings [28].

Also, the scale of expected sensornet deployments precludes manual configuration; several aspects of network and system self-configuration are currently active research topics. Important advances have been made in topology self-configuration [26, 6], localization (in accurate ranging using acoustic or ultrasound signals [23, 22, 11]), and in location estimation coordination techniques by which nodes can determine their location [4, 25]), and post-facto low-energy time synchronization [9].

L3: Packet routing: The MAC layers described above allow nodes within radio range to communicate. Packet routing algorithms are required to deliver packets between non-neighboring nodes. Because the identity of individual nodes plays no role in sensornets, packet routing systems based on node identifiers are unsuitable for sensornets. It is expected that sensornets will instead implement geographic routing systems. Such systems are designed to deliver packets to a single node, or to a collection of

nodes, based on their location (rather than their identity). Below we describe several types of geographic routing systems, each with their own communication abstraction and energy costs. In the following, we let n be the number of nodes in the sensornet, and assume the nodes are uniformly dispersed in the sensornet and that the diameter of the sensornet is $O(\sqrt{n})$.

Strongly geographic routing algorithms, like GPSR [16], allow nodes to send to a particular location. To go from one random location to another requires $O(\sqrt{n})$ packet transmissions, which is our (approximate) metric for total energy consumption. *Weakly geographic* routing algorithms like GEAR [31] allows a node to send packets to a region and then distrubute the packet within that region. The transmission costs here are $O(\sqrt{n})$ packet transmission to reach the region and $O(r)$ packet transmissions within the region, where r is the number of nodes in the region.

In addition to geographic routing, two other packet routing primitives are likely to be available in sensor networks. *Scoped flooding* algorithms flood to a limited region around the sending node. The transmission cost is $O(r)$ where r is the number of nodes in the region. *Flooding* sends a packet to the entire sensornet, and requires $O(n)$ packet transmissions.

L4: Local collaborative information processing: As noted previously, event detection sometimes requires synthesizing the results from several different sensors. The algorithms in this class only require collaboration between local nodes; *i.e.,* those that can be reached by a tightly-scoped flood. An example of such an algorithm is that described in [29]. It uses sophisticated signal processing techniques to correlate sound signals in order to spatially locate its source.

L5: Wide-Area data dissemination: These are techniques by which nodes, and users, can access the desired data anywhere in the sensornet. Recall that data, both observations and events, are named. The data dissemination methods we refer to here allow nodes to access data by name across the sensornet. Note that, in contrast, the local collaborative information processing only used data that could be found nearby; these wide-area data dissemination methods are needed to do collaborative processing in the wide area, as we describe below.

The most commonly mentioned wide-area data dissemination technique is directed diffusion [14, 12]. Directed diffusion involves sending requests for named data and allowing those nodes that have such data to respond; the requesting node can, in addition, *reinforce* high-quality data delivery paths. A key feature of directed diffusion is *in-network aggregation*; as the data is being routed to the requestor, it may be aggregated by intermediate nodes. Directed Diffusion is an example of *data-centric routing*; routing decisions are based on the name of the data rather than on the identity of the sending and receiving nodes.

In this paper we propose another data dissemination approach: *data-centric storage*. As described previously, once an event is identified the associated data is stored by name at another sensornet location which is a function of the data's name. Subsequent requests for that data are sent to that particular node. In this paper we present one mechanism for achieving data-centric storage, but there are doubtless many other approaches. In particular, the geographic location system (GLS) used in the GRID system [19] can be extended to provide a similar capability. We will discuss this in more detail in Section 4. The SCOUT sytem might also be used in a similar way [18].

It should be noted that directed diffusion (and most other data-centric routing proposals) do not require any packet forwarding methods other than flooding. In contrast, the data-centric storage proposal we present here requires strongly geographic routing. Thus, the data dissemination method choice may be limited by the nature of the underlying packet routing mechanisms.

L6: Wide-area collaborative information processing: These methods are akin to the local collaborative information processing methods mentioned above, except that the collaborating nodes need not be local. An example of such a collaboration is that required for tracking an object across a sensor field. In this case, scalable collaborative methods must be built on efficient wide-area data-dissemination algorithms. Zhao *et al.* [33] describe a a collaborative tracking application built on top of directed diffusion.

L7: User-level tasking and querying: The highest layer is where users insert their tasking and querying commands. An example of an approach that fits here is work that has been done on defining database semantics for queries on sensor networks [2].

Our purpose in the preceding discussion was twofold, to provide a very cursory overview of related work and to discuss how these various pieces fit together. We used the notion of "layering" as a convenient organizing principle, but we doubt that real sensornets will strictly obey these distinctions.

2.5 Possible Sensornet Approaches

The preceding discussion reviewed related work, but considered the various algorithmic pieces as separate entities. We now discuss how they fit together to provide a coherent sensornet data-gathering service. We start by considering three *canonical* methods, methods that illustrate three very different approaches to sensornet design.

All of the methods we discuss below start by flooding the tasks to the entire sensornet. The tasks are the set of identification instructions, specifying which events to detect, how to detect them, and what actions to take upon detection. This initial flooding incurs a cost of $O(n)$. We assume that the tasks are long-lived (*i.e.,* that the tasking instructions remain in force for long periods of time) and so this initial cost will presumably be dominated by the ensuing data processing.[2]

2.5.1 Canonical Methods

Earlier we described three basic actions nodes could take upon detecting an event. These lead directly to three canonical sensornet methods.

[2]Of course, there are situations where tasks are short-lived; for these, the cost of flooding dominates all other costs, so it won't matter much which of the approaches below are used.

External Storage (ES): Upon detection of events, the relevant data is sent to extenal storage where it can be further processed as needed. This entails a cost of $O(\sqrt{n})$ for each event. There is no cost for external queries since the event information is already external; queries generated by internal nodes in the process of doing event detection will incur a cost of $O(\sqrt{n})$ to reach the external storage.

Local Storage (LS): Event information is stored locally (at the detecting node) upon detection of an event; this incurs no communication costs. Queries are flooded to all nodes at a cost of $O(n)$. Responses are sent back to the source of the query at a cost of $O(\sqrt{n})$.

Data-Centric Storage (DS): Here, after an event is detected the data is stored by name within the sensornet. The communication cost to store the event is $O(\sqrt{n})$. Queries are directed to the node that stores events of that name, which returns a response, both at a cost of $O(\sqrt{n})$.

These three canonical methods have very different cost structures. We will compare these methods analytically in Section 3 and through simulation in Section 5. However, these three approaches do not exhaust the design space. One can certainly combine them to create hybrid methods. Also, there are alternate approaches geared towards slightly different environments. We discuss these hybrid and alternate approaches in the next two subsections.

2.5.2 Hybrid Methods using Data-Centric Storage

Data-centric storage provides an efficient way to locate information about all events of a given type. However, forcing all data into a fixed event structure may be overly rigid for many applications. In these cases data-centric storage could be used to provide assistance for other approaches.

Using Data-Centric Storage for Context: In the course of processing local data, nodes may find it useful to have some context about global parameters. For instance, data-centric storage could give nodes access to the number of other animals sighted when a node is trying to determine if a migration is underway.

Using Data-Centric Storage for Location Guidance: One of the problems with the local-storage approach is that one needs to flood the query in order to find the relevant data. However, data-centric storage could provide information about the location of the event, and subsequent queries could be directed to their location. For instance, all elephant sightings could be stored at a given node. A query to that node would yield the time and location of all such sightings. More detailed follow-on queries could be sent to the relevant locations.

2.5.3 Alternate Methods

These are methods designed to address slightly different problems than what we consider here.

Directed Diffusion with Reinforcement: This approach, described earlier in our discussion of related work, was originally proposed in [14]. Event information is stored locally at a node upon detection. Queries are flooded to all nodes, and nodes with relevant information respond. The source of the query then reinforces the best responses (in terms of latency or relevancy). This reinforcement process will result in a tree of data flowing from selected sensornet nodes, those that are providing the reinforced data, into the query originator. The cost of the flood is $O(n)$, the cost of each responding packet is $O(\sqrt{n})$. While useful in other contexts, this approach is ideally suited to monitoring ongoing events after a query has been issued. The reinforcement procedure allows the query originator to focus in on the data (and routes) that provide the most relevant and timely information.

Geographically Targeted Queries: The canonical methods are designed for cases where one doesn't *a priori* know the event location. However, if one already knows the location of the event through out-of-band techniques, then one can direct queries to that location using strongly or weakly geographic routing methods (see [31]). In this case, data is stored locally and then queries are sent (at cost $O(\sqrt{n})$) to the relevant locations to retrieve the desired data. This avoids the cost of flooding in the canonical local-storage approach, and avoids the cost of storing each event in the canonical data-centric storage approach. Thus, if event locations are known in advance then these approaches are far superior to the canonical approaches (which don't make use of that information).

3 A Relevant Scenario

We now consider a scenario in which we evaluate the three canonical methods. We first describe the scenario and then provide some approximate formulae for the communication costs for the three methods. While we assume that sensornets would provide all three of these methods (and presumably others), the insight from these formulae help to choose which of them are best suited for a particular task.

3.1 Description of Scenario

The scenario we discuss was not selected as necessarily the most natural but as one in which all three candidate canonical mechanisms might be used. There are many other scenarios that are relevant to sensornets, but they aren't situations where these canonical mechanisms are relevant. We now describe our assumptions about this scenario.

In our scenario, we assume that the event locations are not known in advance and occur in random locations. As mentioned above, if the event location is known, then one can direct queries to the relevant nodes and need not use any one of our three canonical methods. Similarly, if all event of a given type occur in the

same place, then once the first event is located all subsequent queries can be directed there.[3]

Moreover, we assume that tasks are long-lived and don't need to be refreshed very often. If sensors are only "turned on" for short periods of time in response to each request, and these tasking requests are flooded, then queries might as well be piggybacked on top of the flooded tasking requests.

In addition, we assume that there is a single access point. Our discussion can easily be extended to cases where there are a few access points. However, if the number of access points is large, then the access path (which is the set of paths to the external world) may be less of a hotspot (since there are so many access points to choose from) and the following analysis would be misleading.

We also assume that the sensornet expends most of its energy *detecting* events rather than providing ongoing *monitoring* of data. As we've observed, event notification can be used to set up a flow of real-time data from some set of sensornet nodes to the access point(s). If the energy consumption is dominated by this monitoring process then it is of less import which event notification method is used.

Lastly, we assume that in some cases users only care about a summary of the events rather than a listing of each one; *e.g.*, one might just want a count of the number of elephants seen rather than a listing of each elephant sighting. We will consider the two cases – summarized or listed – separately in our analysis and simulations.

Our scenario is described by several parameters. We consider a sensornet with n nodes equipped to detect T event types. We let D_{total} denoting the total number of events detected, Q denote the number of event types for which queries are issued, and D_q denote the number of events detected for the types of events queried for. We assume there is no more than one query for each event type, so there are Q queries in total.

3.2 Approximate Communication Costs

The communication costs of the three methods can be approximated using the asymptotic costs for floods ($O(n)$) and direct routing ($O(\sqrt{n})$). Below we present approximations for both the total number of packets and the packets arriving at the access point. We assume that the packet count at the access point is a good estimate of the hotspot usage, since we expect that the access point to be the most heavily used area of the sensornet.

External Storage:
Total: $D_{total}\sqrt{n}$
Hotspot: D_{total}

Local Storage:
Total: $Qn + D_q\sqrt{n}$
Hotspot: $Q + D_q$

[3] Of course, there is a continuum where future events might be spatially correlated with past events. We just consider the extreme case of no correlation; how to handle cases where there is some limited correlation will depend on the details of the situation.

Data-Centric Storage:
Total: $Q\sqrt{n} + D_{total}\sqrt{n} + D_q\sqrt{n}$ (list)
Total: $Q\sqrt{n} + D_{total}\sqrt{n} + Q\sqrt{n}$ (summary)
Hotspot: $Q + D_q$ (list) or $2Q$ (summary)

where (list) indicates a full listing of events is returned (requiring a packet for each event) and (summary) indicates only a summary of events is returned (requiring only one packet).

These calculations support a few relatively obvious points. First, if all other parameters are held fixed, then as n gets large the local storage method incurs the highest total packet count. Second, external storage always incurs a lower total message count than data-centric storage, but the ratio $1 + \frac{Q+D_q}{D_{total}}$ is unlikely to be large if there are many events detected (and, if there is at least one event detected of each type, this ratio is bounded by 3). Third, if $D_q \gg Q$ and events are summarized, then data-centric storage has the lowest load (of all three methods) on the access path. Fourth, if events are listed and $D_{total} \gg D_q$ then data-centric storage and local storage have significantly lower access loads than external storage.

This suggests that data-centric storage is preferable in cases where (a) the sensornet is large, (b) there are many detected events and not all event types are queried, so that $D_{total} \gg \max[D_q, Q]$. This performance advantage is further increased if summaries are used. However, if if the number of events is large compared to the system size, $D_{total} > Q\sqrt{n}$, and event lists (rather than summaries) are used, then local storage may be preferable.

4 Mechanisms for Data-Centric Storage

In this section we describe our proposal for how to support data-centric storage. This is primarily a proof-of-concept design with many aspects that could be significantly improved.

Data-centric storage is basically a distributed hash-table, supporting two basic primitives:

Put(key,value) This stores value (the data) according to the key (which is based on the name of the data).

Get(key) This retrieves whatever value is stored at the key, which is derived from the event name.

We support this interface in two steps; we first use the GPSR geographic routing algorithm for low-level routing. We then use an approach borrowed from the recent family of efficient peer-to-peer lookup algorithms (Chord, CAN, Pastry and Tapestry [27, 24, 8, 32]) to build a distributed hash-table (DHT) on top of GPSR. In this section we first describe GPSR and then how we layer a lookup algorithm on top of GPSR. At the end of this section we also briefly discuss how one can use the GRID Geographic Location Service (GLS) [19] to support a similar interface.

4.1 GPSR

There have been a number of proposals (see [10, 16, 19, 3] for representative examples), primarily in the ad hoc networking

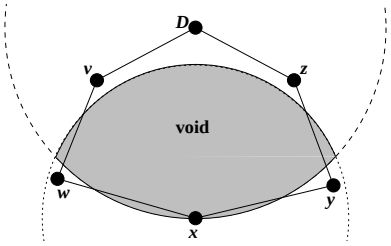

Figure 1: Void Example: x has no neighbor closer to D.

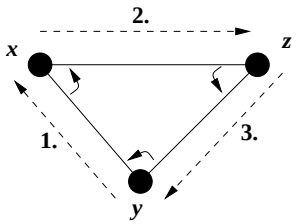

Figure 2: Right-hand Rule Example: Packets travel clockwise around the enclosed region.

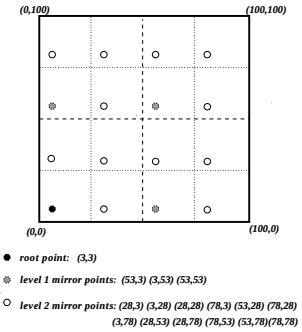

Figure 3: Example of Structured Replication with a 2-level decomposition

arena, for what we call *strongly geographic* routing algorithms. These algorithms take, as the destination, a location not an address; the routing algorithm then delivers packets to the node at that location. All of these algorithms adopt a similar approach. Each node determines its own geographic coordinates (using, for example, GPS) and periodically announces its address and coordinates to its neighbors (*i.e.*, those nodes within its radio range); each node is thus aware of the identities and positions of all its neighbors. Packet headers contain the geographic coordinates of the destination and, whenever possible, packets are forwarded greedily to the neighbor closest to the destination coordinates. When a packet reaches a node that does not know any neighbors closer than itself to the destination, we say that the packet has reached a *dead-end*. Dead-ends are the result of *voids* in the sensornet (see Figure 1), regions with no neighboring nodes in the direction of the destination. The various algorithms [3, 16, 15] adopt different dead-end solutions. In our proposal we adopt a routing algorithm based on Greedy Perimeter Stateless Routing (GPSR) of Karp and Kung [16]. However, we change the semantics of GPSR slightly, so that packets are delivered to the node that is *closest* to the destination location (whereas in GPSR and other similar geographic routing algorithms, packets are dropped if there is no node at the destination location). We now briefly describe our modified version of GPSR; see [16] for the full description of the original GPSR algorithm.

GPSR forwards packets greedily whenever possible. Figure 1 shows an example of greedy forwarding failure; a dead-end caused by a void. The dotted line represents the radio range of x, and the dashed line represents the circle with radius $\overline{xD}$ centered at D. Solid lines show radio connectivity. Here, x can reach D along paths through either of its neighbors w or y, but cannot do so greedily, as w and y are farther from D than x. Intuitively, x seeks to send packets for D *around* the shaded void. Figure 2 shows an example of the *right-hand rule*, a long-known graph traversal that walks a cycle in a graph. Each node to receive a packet forwards the packet on the next link counterclockwise about itself from the ingress link. In this example, when a packet arrives at x from y, x sweeps counterclockwise from $\overline{xy}$ to find $\overline{xz}$. Then z does similarly, and forwards to y. The right-hand rule can circumnavigate the void in Figure 1, and recover from the dead-end at x. For the right-hand rule to take tours of enclosed cycles as shown, the network graph must be *planar*—there can be no crossing edges in it. GPSR constructs a planar subgraph of the network graph, either the Relative Neighborhood Graph (RNG) or Gabriel Graph (GG). These are both connected if the wireless network graph is connected, and can be computed distributedly by nodes, using only knowledge of their immediate neighbors' positions. When a packet dead-ends, GPSR places it into "perimeter" forwarding mode, and forwards it on the edges of a planar subgraph around the perimeter of the void, using the right-hand rule. The first node and edge along which a packet is forwarded in perimeter mode is recorded in the packet header, and greedy forwarding resumes when the packet reaches a a node closer to the destination than the point at which it entered perimeter mode. If a packet traverses the entire perimeter and returns to its entry point, then that node must be the closest node to the destination. The node then accepts that packet as intended for it. Thus, given destination coordinates, GPSR uses a combination of greedy and, if required, perimeter mode forwarding to reach the node closest to the destination coordinates. It is this, and only this, property of GPSR that we use in what follows; we could employ any other scalable routing algorithm that consistently delivers packets to a deterministic node when routing to a particular destination location.

4.2 Distributed Hash-Table (DHT) over GPSR

The basic idea we use here is simple, and is borrowed from the recent peer-to-peer lookup service literature [27, 24, 8, 32]. Given an event name, we hash that name into a key. This key is a location somewhere in the sensornet. Thus, the hash function must be chosen with knowledge of the boundaries of the sensornet.[4] The *put(key,value)* command sends a packet with the given payload into the sensornet routed towards the location *key*. Our (slightly) modified form of GPSR will forward the packet to the sensornet node closest to this location, where the data can be stored; we will call this closest node the *home* node for an event. Similarly,

[4]The boundaries need not be exact; it is only important to avoid hashing to locations outside of the sensornet, as that will require messages to travel around the entire perimeter of the sensornet.

a *get(key)* command is forwarded to the sensornet closest to the *key* location, and that node then returns a packet to the source of the query with the corresponding data.

This approach is sufficient if sensornet nodes are completely stationary and reliable, and the number of events stored is manageable. However, to improve data availability we now extend this approach to deal with failure, mobility, and large numbers of events. We present two extensions: *local refreshes and replication* to deal with robustness and *structured replication* to deal with scalability.

4.2.1 Robustness Extension

While data-centric storage is not intended for sensornets with highly mobile or unreliable nodes, the methods we employ should be resilient to some degree of failures, arrivals, and mobility. The local refresh and replication extensions we describe here are completely GPSR specific and alternate schemes are entirely conceivable. Our description here is at a somewhat high level; the detailed pseudocode is shown in Figure 4.

A home node for an event type periodically routes a *refresh* message to the event location. Regular GPSR routing returns this refresh message to the home node traversing the perimeter along the way. If the dynamics of the situation (whether mobility or new nodes) has resulted in a different node being closer to *key* than the current home node, then this new closer node will become the new home node; in particular, if node A receives a refresh initiated by a home node farther away than node A, then node A does not forward the refresh but instead initiates one of its own. A simple timer-based algorithm ensures that in case the home node dies, the new home node automatically starts generating refresh messages. While this adjusts to dynamics in the perimeters, it does not protect against lost data due to node failure. This is accomplished by *local replication:* all nodes that receive a refresh cache the data contained in it. Even though multiple nodes on a perimeter cache the same events, typically only the home node sends refreshes. It is the refresh from the home node itself that suppresses refreshes from other nodes on the same perimeter, as shown in the pseudocode. Absent the arrival of refreshes, a node will begin to generate refreshes. This process ends with one node on a perimeter becoming the new home node for that event. Our simulations verify this behavior. If these cached items are not refreshed within a certain time-out period, the data are deleted. In addition, when a refresh message, or a query, passes through a node, it merges its own cached data with the data currently in the message (*i.e.*,, if a node has some events that are not listed in the refresh message's data, then this information is appended to the message).

To avoid problems due to packet loss, all *put* and *get* requests are *acked* by the home node. If no *ack* is received, then the initiating node (or user) resends the message.

These modifications improve data availability. They use local replication, which is useful if node failures are randomly distributed, but is of little use if all nodes in an area fail at the same time (*e.g.*, a fire destroys all nodes in a region). Resilience against these *clustered failures* could be provided by storing each event multiple times (using different hash functions).

These techniques assume that the set of neighbors does not change very rapidly. However, if the radio channel characteristics change rapidly in time, then our approach doesn't apply.

4.2.2 Scability Extension

The basic version of data-centric storage has all events of the same type stored in the same place. If an event type has too many events detected, this node could become a hotspot. To deal with this, we use something we call *structured replication* (SR). In SR we augment event names with a hierarchy depth and use a hierarchical decomposition of the key space (similar to that used in GLS). Previously an event name was hashed into a single point in the space; we call that the *root* of an event. Now, for a given root r and a given hierarchy depth d, one can compute $4^d - 1$ mirror images of r; $d = 0$ refers to the original scheme with no mirrors. For example, Figure 3 shows a $d = 2$ decomposition, and the mirror images of the root point (3,3) at every level.

A node that detects an event now stores the event at the mirror closest to its location (this can be easily computed). This reduces the storage costs from $O(\sqrt{n})$ to $O(\sqrt{n}/2^d)$. Queries, however, must now be routed to all the mirror nodes. This is done through a recursive process. Queries are first routed to the root node, then from the root node to the three level 1 mirror image points. Each of these in turn forwards the query to the three level 2 mirror points associated with them. This recursive process is continued until all mirrors are reached. Responses traverse the same path as queries but in the reverse direction – up the hierarchy towards the root. Thus, a single query incurs a routing cost of $O(2^d\sqrt{n})$ as compared to the $O(\sqrt{n})$ for DHTs without mirrors. Hence, for an event I with D_i detected instances and Q_i queries the total message cost of storing and retrieving this event information is approximately: $O(Q_i 2^d\sqrt{n} + D_i(\sqrt{n}/2^d))$ Thus, the use of mirrors reduces the cost of storage but increases the cost of queries. This presents an intermediate solution between the local storage canonical method, where storage was free but queries expensive, and the original data-centric storage algorithm, where both were of moderate cost.[5] We expect that mirrors will be useful for frequently detected events. Note that the depth of the hierarchy (d) can, and indeed should, be different for different event types. [6]

4.3 Grid GLS

We end by mentioning how the Grid Geographic Location Service [19] could be used to provide a similar service. GLS was not intended for this purpose, and so any faults with the following proposal lie with us and not the designers of GLS.

Geographic routing delivers packets to locations, not addresses; thus, to communicate with a particular node one needs a mechanism to map node identifies to their geographic location. GLS is a scalable location service that performs this mapping.

[5] When we choose d such that $2^d = \sqrt{n}$, we reproduce the local store cost structure.

[6] One might, for example, encode the hierarchy level in the event name so that d is globally known for each event type.

The location database is distributed across the nodes; each node acting as the location server for a small number of other nodes. The crux of the problem is that nodes must be able to find these location server without knowing their geograph location. Grid achieves this with a novel algorithm that uses a predefined hierarchical decomposition of the geographic space into nested grids and a predefined ordering of node identifiers. Thus, what GLS enables is routing to node *identifiers*. Moreover, an attempt to route to an identifier Y for which no node exists, terminates at the node with identifier closest to Y as per the predefined ordering of identifiers. Thus, we could use GLS to provide the DHT interface by hashing keys to the node address space. The main drawback with the above approach is that supporting the DHT interface requires the location database to be built and maintained. While for GLS the maintenance of this database was the original goal, for our sensornet application the location database is merely an unnecessary level of indirection. Thus, we chose to directly map event names to locations (rather than to a server which would then reveal the location).

5 Simulation Results

In Section 3, we argued qualitatively about the scaling behavior of the different dissemination mechanisms and their applicability under different scenarios; in Section 4 we proposed one mechanism for data-centric storage based on GPSR. In this section, we first evaluate the performance of our proposed mechanism (Section 5.1) in *ns-2* simulations of relatively small systems of between 50 and 200 nodes. These simulations include detailed models of a wireless network's MAC and physical layer. While our system was primarily designed for environments with fairly stable (not failing) and static (not moving) nodes, we nonetheless would like our design to be robust against small amounts of failure and mobility. To investigate the robustness of our DHT system in harsh wireless environments, we have included simulations with both failing nodes and mobile nodes.

After confirming the viability of our design, we then (Section 5.2) verify our comparative reasoning in Section 3 with simulations of much larger-scale systems of up to 10^5 nodes that, in the interest of computational tractability, do not model radio details.

5.1 Small-Scale Networks, with Wireless Details

We implemented our GPSR-based DHT in *ns-2* [20], which supports detailed simulation of mobile, wireless networks using IEEE 802.11 radios. In these simulations, we seek to demonstrate the robustness of our DHT system on real radios and dynamic topologies, where node failures and mobility cause changes in nodes' neighbors, and changes in the node closest to a *key*'s hashed coordinates.

By modeling the full 802.11 MAC layer and physical layer, *ns-2* allows evaluation of a system's performance on a bandwidth-limited, contention-prone wireless medium. Our simulations use a modified 802.11 radio with a 40 m radio range, rather than

```
_recvInsert(H, I, S)
  // node H receives an insert for event I from node S
    _localStore(I); // Store event information locally
    _send(INSERT-ACK,P_s); // tell node S we received insert
    _routeRefresh(I,P_i) // route refresh for I to P_i
    _schedRefresh(I,(now+2T)) // schedule next refresh for 2T later
    _schedDeleteTimer(I,(now+3T)); // set I's delete timer for 3T later

_recvRefresh(X, I, Y)
  // node X receives refresh for event I from node Y
    _updateStore(I); // store event info from refresh message locally
    _schedDeleteTimer(I,(now+3T)); // reset delete timer for 3T later
    if (X == Y) // refresh looped on perimeter; X is home node
        _schedRefresh(I,(now + T)); // schedule next refresh for T later
                // T rather than 2T because X is the current home node
        return;
    if (distance(P_x, P_i) < distance(P_y, P_i))
      // X is closer to the event position than Y
        drop incoming refresh;
        routeRefresh(I,P_i);
        _schedRefresh(I,(now + 2T)); // 2T rather than T as we're not
                    // sure yet that X is the current home node
        return;
    if (distance(P_x, P_i) > distance(P_y, P_i))
      // X appends any additional info it holds for event I to
      // the refresh message and forwards it on
      _appendRefreshMsg(I);
      _fwdRefresh(I,P_i);
      _schedRefresh(I,(now + 2T)); reset refresh timer for 2T later
      return;

_refreshTimeout(X, I) // X's refresh timer for event I expires
    _routeRefresh(I);
    _schedRefresh(I,(now + 2T));

_deleteTimeout(X, I) // X's delete timer for event I expires
    _localDelete(I); delete all entries for I from local store
    _cancelTimers(I);

_newNode(X, Y) // X detects new node Y
    foreach (event I in local store)
        if ((distance(P_x, P_i) > distance(P_y, P_i)) AND //
                    (X closest of its neighbors to Y))
            _localInsert(I,Y); X sends Y info for event I
            _schedRefresh(I,(now + 2T));

_recvLocalInsert(Y, I) // new node Y receives info for event I
    _localStore(I);
    _schedRefresh(I,(now + T)); Y might be the new home node
```

Figure 4: Pseudo-code for local replication and refreshes. For an event I, we use P_i to denote the corresponding event position. Likewise, for node X, P_x denotes the node's position. All timers have a small randomized component.

Node Density	1 node / 256 m^2
Radio Range	40 m
GPSR Beacon Interval	1 s
GPSR Beacon Expiration	4.5 s
Planarization	GG
Mobility Rate	0, 0.1, 1 m/s
Number of Nodes	50, 100, 150, 200
Simulation Time	300 s
Query Generation Rate	2 qps
Query Start Time	42 s
Refresh Interval	10 s
Event Types	20
Events Detected	10 / type

Table 1: Simulation parameters in *ns-2* simulations.

the 250 m radio range of IEEE-compliant hardware; this choice mirrors that made in the evaluation of Directed Diffusion in [14], in the interest of using parameters close to those found in sensor radios.

In all our *ns-2* simulations, there is a single querying node placed in the upper-left corner of the simulated region. This node represents the access point where queries enter the sensor network. At the start of a simulation, all events are inserted into the DHT *once*, by sensors chosen uniformly at random; these are the sensors that measured the inserted events. Queries are acknowledged and retried until they succeed. At time 42 s, to allow the DHT to stabilize, the querying node begins generating queries at a rate of 2 qps, including both new and retransmitted queries.

In all simulations, we employ the *join optimization*, (routine newNode in Figure 4) which improves performance on dynamic topologies. When a node *A* senses a new neighbor *B*, *A* sends *B* all those event entries from its local database for which *B* is closer to the event destination than *A*, and for which *A* is the closest of its neighbors to that event destination. This optimization trades off increased communication for improved robustness when nodes fail or move.

Table 1 shows the parameters we used in our *ns-2* simulations. We present results that are averaged over multiple simulations; in all cases, the variances of these runs are reasonable. Note that node density remains constant in our simulations; as we increase the number of nodes, we scale the region size such that node density does not change.[7]

In measuring the DHT's performance, we are concerned with the *availability* of the data stored in the DHT to queriers, and the *load* placed on nodes participating in the DHT, both in communication and storage of events. To measure availability, we propose the metric of *success rate*, measured after all events have been inserted into the DHT: for a workload of queries, we compute the mean over all queries of the fraction of events returned in each response, divided by the total number of events known to have been stored in the network for that *key*. Because insertions and queries are both acknowledged, this measurement focuses mainly on the ability of the DHT to hold data written to it.

[7] We do not investigate varying node densities in this work. Karp's thesis demonstrates the efficacy of perimeter-mode forwarding on both dense and sparse networks [17].

Number of Nodes	Success Rate (%)	Max Storage	Avg Storage	Total Msgs	Refresh Msgs
50	100%	47.2	40.7	10.2	4.4
100	100%	11.9	10.0	2.6	1.1
150	99.8%	7.2	5.9	1.6	0.72
200	100%	5.8	4.6	1.2	0.53

Table 2: Performance of DHT on Static Networks. Results are the means of three simulations.

To measure the storage load on nodes, we examine the maximum number of events stored at any node, to capture the worst-case required storage; and the mean number of events stored across all nodes in the network, to capture typical storage requirements. We measure the communication load on nodes by counting the mean number of messages forwarded by a node in a refresh interval, and the mean number of refresh messages forwarded by a node in a refresh interval; these message counts are averaged across all nodes and refresh intervals in a simulation.

5.1.1 Stable and Static Nodes

As one would expect, on static networks, where the topology doesn't change, DHT offers very nearly perfect availability of stored events. At all network scales, essentially all queries are answered with all events stored in the network. As the system scales in increasing number of nodes, the unchanging number of events are dispersed among a wider population of nodes, and thus both the mean and maximum state requirements per node decrease. Similarly, dispersion reduces the count of the mean number of forwarded refresh messages; fewer nodes are on perimeters about a point to which a *(key, value)* pair hashes, and so a smaller fraction of nodes receives refresh messages for forwarding.

5.1.2 Static but Failing Nodes

We now demonstrate that DHT is robust in the presence of node failures, despite the topology changes that result. All the results we present in this section are for networks of 100 nodes.

Table 3 shows the performance of DHT under a failure model where a configured fraction of nodes selected uniformly at random alternate between failing and restarting. When a node fails, it loses the contents of its database; it only reacquires its database contents upon returning to operation and receiving refreshes from neigbhors. In these results, a node selected as unreliable remains up for a period selected uniformly at random in $[0, 120]$ s, then goes down for a period uniformly chosen in $[0, 60]$. We denote by f the fraction of nodes that remain up for the entire simulation.

As one would expect, the success rate decreases as f does. But the decrease is slight, until *all* nodes cycle between available and unavailable, at $f = 0$. The deterioration in the success rate is caused by events that were not saved by the refresh mechanism when the node holding them failed. Analysis of the simulation logs reveals that the vast majority of queries and responses reach

f	Success Rate (%)	Max Storage	Avg Storage	Total Msgs	Refresh Msgs
0	83.3%	25.4	8.8	3.2	1.6
0.2	94.2%	24.9	10.3	3.4	1.8
0.4	97.3%	22.6	10.7	3.4	1.8
0.6	98.6%	17.4	10.3	3.1	1.6
0.8	99.7%	14.0	10.1	3.1	1.5
1.0	100%	16.2	14.5	3.9	1.6

Table 3: Performance of DHT. Stationary nodes, varied fraction of nodes alternate between up and down states. Results are the means of eight simulations.

Up/Down Time (s)	Success Rate (%)	Max Storage	Avg Storage	Total Msgs	Refresh Msgs
60/30	75.1%	18.6	6.0	2.9	0.93
120/60	84.7%	29.6	9.8	3.5	1.8
240/120	94.7%	45.9	15.2	4.7	3.1
480/240	95.7%	53.2	17.5	5.3	3.7

Table 4: Performance of DHT. Stationary nodes, all alternate between up and down states of varied lengths. Results are the means of four simulations.

their destination successfully in a single transmission. Note that the maximum number of events stored at a node *decreases* as more nodes become reliable, while the mean number of events stored across all nodes *increases*; these trends reflect the increased uniformity of the distribution of events across nodes, as the number of simultaneously available nodes increases.

Table 4 shows the performance of DHT where $f = 0$ (that is, where all nodes fail and restart repeatedly). Here, we vary the periods that nodes remain up and down. For an up/down time value of x/y, a node remains up for a period chosen uniformly in $[0,x]$, and remains down for a period chosen uniformly in $[0,y]$. Simulation times for this group of simulations *only* are *not* 300 s; we scale the simulation time linearly with the up/down time; each simulation lasts five times the length of a down time interval.

When nodes transition between up and down more frequently, the ability of the DHT to hold events is stressed more heavily, as the node closest to an event's destination position changes more frequently. The success rate decreases very gradually at first, but progressively more noticeably as the up/down periods shorten. The maximum and average storage figures in these cases reflect that events *disappear* from the DHT when the join optimization and refreshes fail to keep events alive in the DHT.

5.1.3 Stable but Mobile Nodes

Table 5 shows how DHT performs on a mobile sensor network of 100 nodes. In these simulations, nodes move using the random waypoint model [20]; that is, in discrete steps, each to a point chosen uniformly at random, at a rate chosen uniformly at random in $(0,M)$ m/s, where M is the maximum motion rate. They pause 60 s between motion steps. In these simulations, we use a timer to cause GPSR to replanarize once every two seconds, which costs no communication; only computation within a sensor node.

Motion Rate (m/s)	Success Rate (%)	Max Storage	Avg Storage	Total Msgs	Refresh Msgs
0.1	96.8%	18.6	10.4	19.2	1.45
1	96.3%	52.2	22.5	17.4	4.10

Table 5: Performance of DHT on mobile networks. 0.1 and 1 m/s mobility. Results are the means of four runs for the 0.1 m/s case, and twelve runs for the 1 m/s case.

While our main thrust is *not* to address DHTs for mobile sensor networks, we nonetheless stress that under mobility, DHT continues to offer robust persistence for stored events, as demonstrated by the 96+% success rates in Table 5. The cost of this robustness is in communication—note the greater number of messages forwarded by DHT in the mobile scenarios, *vs.* in the non-mobile ones. Under mobility, GPSR's perimeters change, and it's possible for a packet walking a perimeter that changes underfoot to loop, until the packet exhausts its TTL in hops [17]. We limit the TTL on refresh messages to ten hops in the mobile simulations; they need not all walk the intended perimeter for refreshes to function properly, and the cost in congestion to the network of forwarding them on far longer tours is significant. In a more general implementation of DHT, a node can dynamically determine the appropriate TTL to use, by periodically sending a refresh with a small TTL, and expanding the TTL until the refresh returns successfully. In these results, we elide this implementation step, and fix the TTL at ten hops for refreshes. This value is longer than the typical perimeter for the network density we simulate.

5.1.4 Discussion

Our GPSR-based design appears to work well in sensornets with stable and static nodes. Such settings are the primary target of our design, but of course failures (and movement, in some cases) are invetable, and thus we are interested in the robustness of our design against these factors. In our various robustness tests we subject our design to very harsh environments. Our most generous run with failing nodes had mean cycle-times on the order of minutes, far worse than we hope for most projected sensornet systems. And yet, as long as the fraction of failing nodes wasn't overly high, or the cycle-times were tens of minutes, the system performed reasonably well. Similary, the extent and rate of movement in the mobile node case was quite high; nodes rested only a minute between movements, and the movements were large excursion (half the size of the sensornet, on average) not slight adjustments. Here too the performance was reasonably good.

5.2 Comparative Study

The detailed ns simulations were intended to verify the correct functioning of the low-level aspects of the data-centric storage mechanism. However, we were limited to system sizes on the

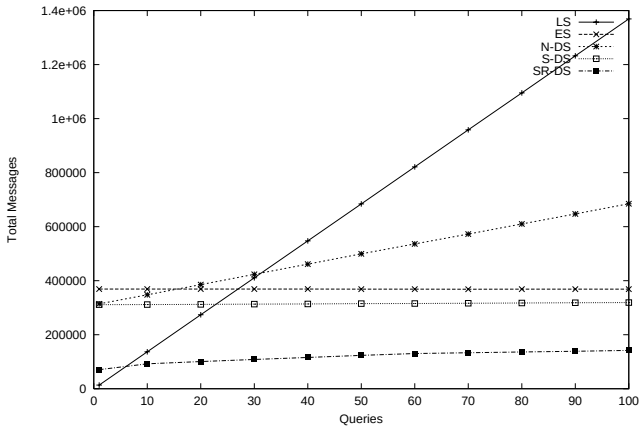

Figure 5: *Total Messages, varying Q.*

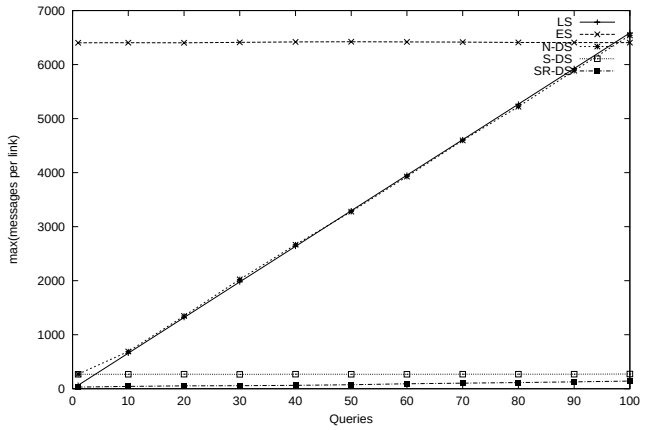

Figure 6: *Hotspot Messages, varying Q.*

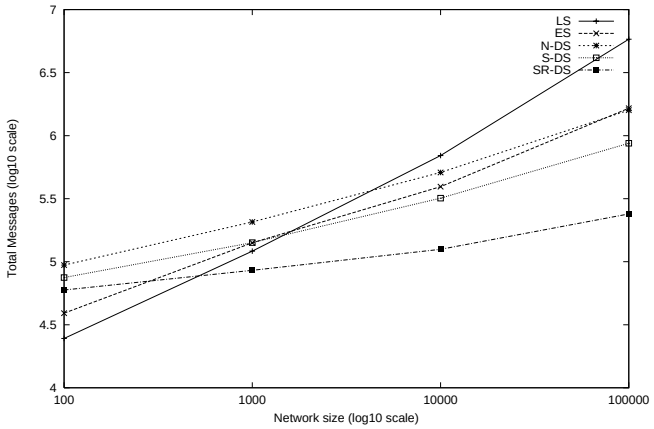

Figure 7: *Total Messages, varying n.*

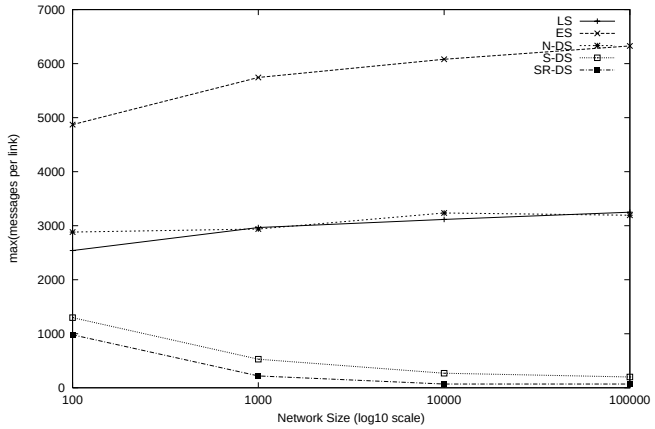

Figure 8: *Hotspot Messages, varying n.*

order of 100 nodes. In this section we use a less-detailed simulations to compare the three canonical mechanisms – external storage (ES), local storage (LS), and data-centric storage (DS) – in much larger systems. We built a special purpose simulator that assumes (1) nodes are stable and stationary and (2) packet delivery to neighboring nodes is instantaneous and error-free. This simulator thus faithfully represent the data paths in the ideal versions of the various canonical mechanisms. We use this simulator to examine the number and pattern of packet transmissions (as a measure of energy consumption) as the size and nature of the system varies. This simulator clearly does not allow us to investigate the low-level details of the wireless environment (MAC level behavior, packet loss, node dynamics, etc.) which were the subject of the previous ns simulations, but nonetheless we feel it gives a good picture of the *relative* performance of the canonical data dissemination algorithms.

We use two metrics to evaluate the performance: the total number of packets generated and the hotspot usage (which is the maximum number of packets to cross any single link). We don't measure latency, as that is approximately the same across the algorithms. Moreover, we assume that all other factors (such as the fidelity of the data) are held fixed across the various algorithms.

The relevant system parameters are:

- n, the number of nodes in the system,
- T, the number of event types
- Q, the number of event types queried for
- D_i, the number of detected events of event type i

In this section we set $T = 100$ and $D_i = 100$ for all i, and vary n and Q. We present two basic tests. In test #1 we hold n fixed ($n = 10000$) and vary Q. In test #2 we set $Q = 50$ and vary n; for reasons we gave in Section 5.1, we hold the system density fixed and increase the sensornet size as we increase n. All these results are averaged across ten different topologies, and ten runs on each topology. In each of these tests, we show the results ES, LS, and the following three versions of DS:

Normal DS (N-DS): A query returns a separate message for each detected event

Summarized DS (S-DS): A query returns a single message regardless of the number of detected events

Structured Replication DS (SR-DS): We assume an optimal level of structured replication (as described in Section 4.2.1) to provide a lower bound. We assume summarization in this case.[8]

Test #1: Varying Q The results from varying Q are shown in Figures 5 and 6. For low Q, LS has low total and hotspot usage, but both quantities increase linearly with Q, making LS a poor choice for systems with many queries. External storage has a very high hotspot load and a medium level of total usage, both independent of Q. Both variants of DS that use summarization have low total and hotspot usage, but note that structured replication in SR-DS is able to bring the total usage down significantly. The hotspot and total usage of DS without summarization (N-DS) increases linearly with Q, but the slope of the total usage is much lower than that of LS (but has a higher offset). These results suggest that for low Q all methods but ES perform reasonably well, with LS and SR-DS being the best. For high Q, SR-DS is the clearly superior choice, followed by S-DS. If summarization is not allowed, then the choice is between N-DS (which has lower hotspot usage) and ES (which has lower total usage).

Test #2: Varying n The results from varying n are shown in Figures 7 and 8. All of the methods have reasonably similar behavior for total usage, but LS starts off (at low n) with the lowest value, and ends up (at high n) with the highest value. S-DS and SR-DS have the lowest hotspot usage by far, but among methods without summarization DS and LS have similar performance. ES has the worst hotspot load. Thus, at all but teh lowest values of n (lower than around $n = 1000$) the DS variants are the superior choices. Recall that these simulations were run with $Q = 50$, and so these conclusions are similar to those in test #1 above.

These performance results are remarkably consistent with the approximate formulae presented in Section 3.2; the only significant deviations coming from cases where the hotspot usage was not occuring at the access point. These simulations, while idealizing the low-level wireless behavior, were true packet-level simulations of these various algorithms in systems as large as $n = 100,000$. This gives us confidence that all three of these canonical methods scale in the ways suggested by the approximate formulae. Those formulae suggested that DS was particularly appropriate as system sizes grew and the number of events is far greater than the number returned in queries (either because not all event types are queried for, or because summarization is used).

We've run additional simulations with various other parameter values, and the results are consistent with these conclusions. Nonetheless, the simulations we've done, and indeed the basic scenario we considered, covers only a small portion of the space of sensornet scenarios. Hence, we claim only that data-centric storage is of use in some reasonably plausible scenarios we've explored here.

[8] We could have looked at structured replication without summarization, but left it out in interests of space (and clutter in the graphs).

6 Discussion

We believe future sensornets will incorporate many different data dissemination mechanisms and users will choose among them based on the particular circumstances. This paper proposed data-centric storage as one possible approach to wide-area data dissemination and identified those circumstances where data-centric storage may be relevant.

The data-centric storage mechanism we presented, while still preliminary and intended mainly as a proof-of-concept, appears to work well over a range of conditions. However, it clearly fails if the node failure rate is extremely high, or if nodes are significantly mobile. In addition, our method will probably not deal well with rapid variations in radio channel characteristics.

In terms of the general approach to data dissemination, we recognize that data-centric storage is irrelevant if event locations are known beforehand. Moreover, one of the other canonical methods is preferable if the number of nodes is small, or the cost of flooding queries is outweighed by the streaming of real-time data, or the number of events is a factor of $\sqrt{n}$ greater than the number of queries.

However, when we are in a context where these adverse conditions do not hold, data-centric storage may be quite useful, in that it enables efficient queries for classes of events. Moreover, it can be a useful component in a more general data dissemination system. For instance, it may allow users to more easily build a complex web of event definitions, where the detection of some higher-level events (*e.g.*, animal migrations) are dependent on the detection of lower-level events (*e.g.*, animal sightings). In addition, it may provide helpful global context for evaluating local data; *e.g.*, a temperature sensor could compare its temperature to the average temperature in the sensornet to help detect fires and other local temperature spikes.

References

[1] W. Adjie-Winoto, E. Schwartz, and H. Balakrishnan. The Design and Implementation of an Intentional Naming System In *Proceedings of the Symposium on Operating Systems Principles*, pages 186–201, Charleston, South Caroline, December 1999. ACM.

[2] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the Physical World. October 2000.

[3] Bose, P., Morin, P, Stojmenović, I., and Urrutia, J., Routing with guaranteed delivery in *ad hoc* wireless networks. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DialM '99), Aug. 1999.

[4] N. Bulusu, J. Heidemann, and D. Estrin. GPS-Less Low Cost Outdoor Localization for Wireless Sensor Networks. October 2000.

[5] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology In 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, Costa Rica, April 2001.

[6] A. Cerpa and D. Estrin. Adaptive Self-Configuring Sensor Network Topologies. In *To appear, Proceedings of IEEE Infocom*, 2002.

[7] Defense Advanced Research Projects Agency. Sensor Information Technology `http:www.darpa.mil/ito/ /research/sensit`

[8] DRUSCHEL, P., AND ROWSTRON, A. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)W* (Nov 2001).

[9] J. Elson and D. Estrin. Time Synchronization in Wireless Sensor Networks. In *Proceedings of the IPDPS Workshop on Parallel and Distributed Computing Issues for Wireless and Mobile Systems*, 2001.

[10] Finn, G.G., Routing and addressing problems in large metropolitan-scale internetworks. Tech. Rep. ISI/RR-87-180, Information Sciences Institute, Mar. 1987.

[11] L. Girod and D. Estrin. Robust Range Estimation Using Acoustic and Multimodal Sensing. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems*, 2001.

[12] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM.

[13] J. Hill, R. Szewcyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[15] Karp, B., Greedy perimeter state routing. Invited seminar at the USC/Information Sciences Institute, July 1998.

[16] B. Karp and H. Kung. Greedy Perimeter Stateless Routing. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[17] B. Karp. *Geographic Routing for Wireless Networks*. Ph.D. Dissertation, Division of Engingeering and Applied Sciences, Harvard University, 2000.

[18] S. Kumar, C. Alaettinoglu, and D. Estrin. SCalable Object-tracking through Unattended Techniques (SCOUT) In *Proceedings of the 8th International Conference on Network Protocols(ICNP)*, Osaka, Japan, November 2000.

[19] Li, J., Jannotti, J., DeCouto, D., Karger, D., and Morris, R., A scalable location service for geographic ad-hoc routing. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)* (Boston, MA, USA, Aug. 2000).

[20] S. McCanne and S. Floyd. *ns* Network Simulator. `http://www.isi.edu/nsnam/ns/`

[21] G. Pottie and W. Kaiser. Wireless Sensor Networks. *Communications of the ACM*, 2000.

[22] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location Support System. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[23] N. Priyantha, A. Liu, H. Balakrishnan, and S. Teller. The Cricket Compass for Context-Aware Mobile Applications. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2001)*, 2001.

[24] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proc. ACM SIGCOMM* (San Diego, CA, August 2001), pp. 161–172.

[25] A. Savvides, C.-C. Han, and M. B. Srivastava. Dynamic Fine-Grain Localization in Ad-Hoc Networks of Sensors. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2001)*, 2001.

[26] C. Schurgers, V. Tsiatsis, and M. B. Srivastava. STEM: Topology Management for Energy Efficient Sensor Networks. preprint, 2001.

[27] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference* (San Diego, California, August 2001).

[28] A. Woo and D. Culler. A Transmission Control Scheme for Media Access In Sensor Networks In Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2001), 2001.

[29] K. Yao, R. Hudson, C. Reed, D. Chen, and F. Lorenzelli. Blind Beamforming on a Randomly Distributed Sensor Array. In *IEEE Journal in Selected Areas in Communication*, October 1998.

[30] W. Ye, J. Heidemann, and D. Estrin. An Energy Efficient MAC Protocol for Wireless Sensor Networks. In *To appear, Proceedings of IEEE Infocom*, 2002.

[31] Y. Yu, D. Estrin, and R. Govindan. Geographical and Energy-Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks. UCLA Computer Science Department Technical Report, UCLA-CSD TR-01-0023, May 2001.

[32] ZHAO, B. Y., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.

[33] F. Zhao, J. Shin, and J. Reich. Information-Driven Dynamic Sensor Collaboration for Tracking Applications. In *IEEE Signal Processing Magazine*, March 2002.