GHT: A Geographic Hash Table for Data-Centric Storage

Sylvia Ratnasamy and Brad Karp
ICIR/ICSI, Berkeley, CA 94704
{sylviar,bkarp}@icsi.berkeley.edu

Yin Li and Fang Yu
Berkeley EECS, Berkeley, CA 94704
{yinli,fyu}@eecs.berkeley.edu

Deborah Estrin
UCLA Comp. Sci., LA, CA 90095
destrin@cs.ucla.edu

Ramesh Govindan
USC Comp. Sci., LA, CA 90089
ramesh@usc.edu

Scott Shenker
ICIR/ICSI, Berkeley, CA 94704
shenker@icsi.berkeley.edu

ABSTRACT

Making effective use of the vast amounts of data gathered by large-scale sensor networks will require scalable, self-organizing, and energy-efficient data dissemination algorithms. Previous work has identified data-centric routing as one such method. In an associated position paper [23], we argue that a companion method, data-centric storage (DCS), is also a useful approach. Under DCS, sensed data are stored at a node determined by the name associated with the sensed data.

In this paper, we describe GHT, a Geographic Hash Table system for DCS on sensornets. GHT hashes keys into geographic coordinates, and stores a key-value pair at the sensor node geographically nearest the hash of its key. The system replicates stored data locally to ensure persistence when nodes fail. It uses an efficient consistency protocol to ensure that key-value pairs are stored at the appropriate nodes after topological changes. And it distributes load throughout the network using a geographic hierarchy. We evaluate the performance of GHT as a DCS system in simulation against two other dissemination approaches. Our results demonstrate that GHT is the preferable approach for the application workloads predicted in [23], offers high data availability, and scales to large sensornet deployments, even when nodes fail or are mobile.

Categories and Subject Descriptors:
H.3.4 [**Systems and Software**]: Distributed Systems
General Terms:
Algorithms, Design, Performance

1. INTRODUCTION

A sensornet is a distributed sensing network comprised of a large number of small devices, each with some computational, storage and communication capability. Such networks can operate in an unattended mode to record detailed information about their surroundings. They are thus well suited to applications such as location tracking and habitat monitoring [4, 18]. As these networks scale in size, so will the amount of data they make available. The great volume of these data and the fact that they are spread across the entire sensornet create the need for data-dissemination techniques capable of extracting relevant data from within the sensornet. Moreover, communication between 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.

The utility of a sensornet derives primarily from the data it gathers; the identity of the individual sensor node that records the data tends to be less relevant. Accordingly, sensornet researchers have argued for communication abstractions that are *data-centric*. Under this model, data are "named" and communication abstractions refer to these names rather than to node network addresses [1, 9]. The directed diffusion [10] data-centric routing scheme has been shown to be an energy-efficient data dissemination method for sensornet environments. In an associated position paper [23], we suggest that a companion method, data-centric storage (DCS), will also be useful. Under the DCS approach, the particular node that stores a given data object is determined by the object's name. Hence all data with the same general name (*e.g.,* "elephant sightings") will be stored at the same sensornet node (not necessarily the node that originally gathered the data). The advantage of DCS then is that queries for data with a particular name can be sent directly to the node storing these named data, thereby avoiding the query flooding typically required in data-centric routing proposals.

This paper outlines what we believe are three canonical dissemination methods, one of which is data-centric storage. The three methods have very different performance characteristics. Which one is appropriate for a particular setting will depend on the nature of the sensornet and its use. Consequently, we stress that our point 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 according to the task at hand.

This paper serves two aims. Our first is to identify the circumstances where DCS is the preferred dissemination method. In a related position paper [23], we lay out the context for this comparative study with a lengthy discussion of sensornet dissemination algorithms and the settings in which they might be used. For completeness, we begin our paper with a brief review of this discussion. This review also provides the needed context for the later comparative simulations.

Our second aim is to present design criteria for scalable, robust DCS, and a DCS system that meets those criteria, the Geographic Hash Table (GHT). GHT is inspired by the new generation of Internet-scale Distributed Hash Table (DHT) systems such as Chord, CAN, Pastry, and Tapestry [6, 21, 24, 25]. In these systems, a data object is associated with a key and each node in the system is responsible for storing a certain range of keys. A name-based routing algorithm allows any node in the system to locate the storage node for an arbitrary key. This enables nodes to `put` and `get`

To appear in the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA) 2002.

files based on their key, thereby supporting a hash-table-like interface. GHT uses the GPSR geographic routing algorithm [13] as the underlying routing system to provide a similar hash-table-like functionality in sensornets.

Our paper has 7 sections. We start with a brief discussion of data dissemination in sensornets in Section 2 and elaborate on the DCS problem in Section 3. Section 4 presents the detailed design of GHT which we evaluate in Section 5. We discuss related work in Section 6 and conclude with a short discussion of future work in Section 7.

2. CONTEXT

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 and outline possible approaches to data dissemination in sensornets.

2.1 Assumptions and Metrics

Projected sensornet designs in the literature [5] differ greatly in their characteristics and intended use. In this paper, we focus on a class of sensornets that is most relevant to the data dissemination issues we address.

We consider large-scale sensornets with nodes that are spread out over an area whose approximate geographic boundaries are known to the network operators. We assume that nodes know their geographic location. This can be achieved through the use of GPS or some other approximate but less burdensome localization technique [3, 8, 19, 20, 22]. This assumption is critical for our proposed data-centric storage algorithm. However, we think it is a reasonable assumption because in many cases the sensornet data are useful only if the location of their source is known.

We assume that the sensornet is connected to the outside world through a small number of access points, hence getting data from a sensornet node to the outside world requires routing the data through the sensornet to the access point. This assumption is not required by our DCS mechanism per se but is key to our comparison of the different dissemination mechanisms.

Finally, we assume that energy is a scarce commodity for sensornet nodes [18] 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 consumption:

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

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

2.2 Sensornet Data

In this section, we present our terminology for the different types of sensornet data and describe the operations we envisage will be used to extract relevant data from a sensornet.

2.2.1 *Observations and Events*

We use the term *observations* to refer to the low-level readings from these sensors and the term *events* to refer to pre-defined constellations of low-level observations. For example, detailed temperature and pressure readings might constitute observations, while a particular combination of temperature and pressure observations might define an "elephant-sighting" event.

Typically, the large volume of observations prohibits communicating them directly to the outside world. Events are thus derived by processing the low-level observations within the network and users can then query for events. Once events have been detected, users might want to access the low-level observations related to a particular event. This is easily accommodated by having each event notification include the event's location, so that to gather additional data one need only download the required observations from the relevant sensors.

2.2.2 *Tasks, Actions, and Queries*

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, actions might instruct a node on where to store information for a particular event.

When event information is stored within the sensornet, *queries* are used to retrieve this information from the network. How queries are executed will depend on the actions nodes take upon event detection.

2.3 Approaches to Data Dissemination

Data dissemination starts by flooding the tasks to the entire sensornet. The tasks specify which events to detect, how to detect them, and what actions to take upon detection. Upon detecting an event, there are three basic actions a node can take which lead directly to three canonical data dissemination methods. These three methods have a very different cost structure. In this section, we first describe these canonical methods and then compare their costs analytically; in Section 5 we use simulation to perform a more detailed comparison.

In the discussion that follows, we assume that tasks are long-lived (*i.e.,* that the tasking instructions remain in force for long periods of time) and so the initial cost of issuing tasks is dominated by the ensuing data processing.[1] In evaluating communication costs we use the asymptotic cost of $O(n)$ message transmissions for floods and $O(\sqrt{n})$ for point-to-point routing where n is the number of sensornet nodes.

2.3.1 *Canonical Methods*

When an event occurs, the detecting node has only three options for where the event information can be stored: at external storage outside the sensornet, within the sensornet at the detecting node or within the sensornet at a node other than the detecting node. These three storage actions lead to the following canonical data dissemination methods:

External Storage (ES): Upon detection of events, the relevant data are sent to external storage where they can be further processed as needed. This entails a cost of $O(\sqrt{n})$ for each event (to ship the information to the access point). There is no cost for user queries since the event information is already external.[2]

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

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

[2] If queries can be generated by internal nodes, they will incur a cost of $O(\sqrt{n})$ to reach the external storage.

of $O(n)$. Responses are sent back to the source of the query at a cost of $O(\sqrt{n})$.

Data-Centric Storage (DCS): Here, after an event is detected the data are stored by name (*i.e.,* at a storage node that need not be the same as the detecting node) 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 $O(\sqrt{n})$.

The three approaches above certainly do not exhaust the design space; see [23] for variants and combinations of the above.

2.3.2 Approximate Communication Costs

We now compare the performance of these methods using a simple analytical model. The cost structure for the canonical methods is described by several parameters. We consider a sensornet with n nodes equipped to detect T event types. We let D_{total} denote 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.

In comparing costs, we also consider the case where users only care about a summary of the events rather than a listing of each event; *e.g.,* one might just want a count of the number of elephants seen rather than a listing of each elephant sighting.

We compare costs using approximations for both the total number of packets in the sensornet and the packets arriving at the access point.[3] 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. With this setup, the costs are as follows:

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

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

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, all other parameters being fixed, 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.

We conclude 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 increases further when summaries are used. However, 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.

[3]While we assume a single access point, our discussion extends easily to cases where there are a few access points.

3. THE DCS PROBLEM

We have argued for the utility of a DCS service for sensornets. Now we will define the data-centric storage problem in more detail: the storage abstraction DCS provides, the design goals a robust, scalable DCS system must meet, and our *geographic hashing* approach to DCS architecture that meets these design goals.

3.1 Storage Abstraction

Like the many distributed hash table systems before it [6, 21, 24, 25], DCS provides a *(key, value)*-based associative memory. Events are named with keys. Both the storage of an event and its retrieval are performed using these keys. DCS is naming-agnostic; any naming scheme that distinguishes events that users of the sensornet wish to identify distinctly suffices. The two operations DCS supports are:

Put(k, v) stores v (the observed data) according to the key k, the name of the data.

Get(k) retrieves whatever value is stored associated with key k.

3.2 Design Criteria for Scalable, Robust DCS

The challenge in any design for a DCS system is to meet scalability and robustness criteria despite the system's fundamentally distributed nature. Sensornets represent a particularly challenging environment for a distributed storage system:

Node failures may be routine; exhaustion of battery power and permanent or transient failure in a harsh environment are problems in any realistic sensornet deployment.

Topology changes will be more frequent than on traditional wired networks. Node failures, node mobility, and received signal strength variations in real radio deployments each independently cause neighbor relationships among nodes to change over time.

System scale in nodes may be very great. Sensor nodes may be deployed extremely densely (consider the limit case of smart dust [11]), and may be deployed over a very wide physical region, such that the total number of devices participating in the DCS system may be on the order of 10^6 or more nodes.

Energy constraints will often be severe; nodes will operate from battery power.

These challenges suggest several specific, important design criteria for ensuring scalability and robustness in the distributed storage system we envision:

Persistence: a (k, v) pair stored in the system must remain available to queriers, despite sensor node failures and changes in the sensor network topology.

Consistency: a query for k must be routed correctly to a node where (k, v) pairs are currently stored; if this node changes (*e.g.,* to maintain persistence after a node failure), queries and stored data must choose a new node consistently.

Scaling in database size: as the number of (k, v) pairs stored in the system increases, whether for the same or different ks, storage should not concentrate at any one node.

Scaling in node count: as the number of nodes in the system increases, the system's total storage capacity should increase, and the communication cost of the system should not grow unduly. Nor should any node become a concentration point of communication.

Topological generality: the system should work well on a broad range of network topologies.

3.3 GHT: A Geographic Hash Table

The DCS system architecture we describe in this paper to meet the above-enumerated design criteria is GHT, a *Geographic Hash Table (GHT)*. The core step in GHT is the hashing of a key k into geographic coordinates. Both a **Put()** operation and a **Get()** operation on the same key k hash k to the same location. A key-value pair is stored at a node in the vicinity of the location to which its key hashes. Choosing this node consistently is central to building a GHT. If we assume a perfectly static network topology and a network routing system that can deliver packets to positions, such a GHT will cause storage requests and queries for the same k to be routed to the same node, and will distribute the storage request and query load for distinct k values evenly across the area covered by a network.

The service provided by GHT is similar in character to those offered by other distributed hash table systems [6, 21, 24, 25]. However, as is the case with those systems, much of the nuance to the GHT system design arises specifically to ensure robustness and scalability in the face of the many sorts of failures possible in a distributed system. GHT uses a novel *perimeter refresh protocol* to provide both persistence and consistency when nodes fail or move. This protocol replicates stored data for key k at nodes around the location to which k hashes, and ensures that one node is chosen consistently as the *home node* for that k, so that all storage requests and queries for k can be routed to that node. Yet the protocol is efficient; it typically uses highly local communication, especially on networks where nodes are deployed densely. By hashing keys, GHT spreads storage and communication load between different keys evenly throughout the sensornet. When many events with the same key are stored, GHT avoids creating a hotspot of communication and storage at their shared home node by employing *structured replication*, whereby events that hash to the same home node can be divided among multiple mirrors.

4. ALGORITHMS

We proceed now to describe the algorithms that comprise GHT. GHT is built atop GPSR [12, 13, 14], a geographic routing system for multi-hop wireless networks. After briefly reviewing the features of GPSR's design relevant to GHT, we identify a previously unexploited characteristic of GPSR that allows all packets destined for an arbitrary location (unoccupied by a node) to be routed consistently to the same node in the vicinity of that location. GHT leverages this characteristic to route storage requests and queries for the same key to the same node, despite the ignorance of the hash function that maps keys into locations of the placement of nodes in the network. We then describe algorithms and implementations of the perimeter refresh protocol and structured replication, which allow GHT to achieve the DCS design criteria for scalability and robustness discussed in the previous section.

4.1 GPSR

Under GPSR, packets are routed geographically. All packets are marked with the *positions* of their destinations. All nodes know their own positions, and the positions of the nodes a single hop away from them. Using only this local knowledge, GPSR can route a packet to any connected destination. There are two distinct algorithms GPSR uses for routing: a *greedy* forwarding algorithm [7] that moves packets progressively closer to the destination at each hop, and a *perimeter* forwarding algorithm that forwards packets where greedy forwarding is impossible.

The greedy forwarding rule is simple: a node x forwards a packet to its neighbor y that is closest to the destination D marked in the packet, so long as that neighbor is closer to D than x. Figure 1 shows an example of greedy forwarding; the dotted line represents the radio range of node x, and the dashed line the circle centered at D with radius $\overline{xD}$.

Greedy forwarding fails when no neighbor is closer than x to the destination. Figure 2 shows an example topology for greedy forwarding failure. Here again, the dotted line shows x's radio range and the dashed line the circle centered at D of radius $\overline{xD}$. The solid lines show the links that exist, as dictated by radio range. Note that two paths to D exist, but x cannot forward greedily on either of them because both involve temporarily moving *farther away* than x from the destination.

GPSR recovers from greedy forwarding failure using *perimeter mode*, which amounts to forwarding packets using the *right-hand rule*. Figure 3 demonstrates the right-hand rule: upon arriving on an edge at node x, the packet is forwarded on the next edge counterclockwise about x from the ingress edge. This process causes packets to tour enclosed faces as shown; intuitively, it is useful for circumnavigating regions where greedy forwarding fails, as in Figure 2. GPSR routes perimeter mode packets on a planar subgraph of the network connectivity graph, in which there are no crossing edges. A perimeter is a face of this planar graph. Bose *et al.* [2] also present an algorithm that uses planar network subgraphs to recover from greedy forwarding failure.

GPSR originates packets in greedy mode, but changes them to perimeter mode when no neighbor of the forwarding node is closer to the packet's destination than the forwarding node itself. GPSR returns a perimeter-mode packet to greedy mode when the packet reaches a node closer to the destination than that at which the packet entered perimeter mode (stored in the packet). As will be shown in the next section, our GHT algorithms use perimeter mode in another, novel, way to route packets that refer to the same storage key to the same node.

4.2 The Home Node and Home Perimeter

GPSR was designed for a network model where a sender wishes to transmit packets to a destination node with a known non-geographic address; a sender must map the destination's identifier to its current location using a location database, such as GLS [16]. Under GHT, however, the originator of a **Put()** or **Get()** packet does *not* know the identifier of the node that is the eventual destination of the packet. As sketched in Section 3.3, the originator of a **Put()** or **Get()** for a key k hashes the name k into geographic coordinates that are the destination of the packet for that operation. The hash function is ignorant of the placement of individual nodes in the topology; it merely spreads the different key names evenly across the geographic region where the network is deployed. Thus, it is quite likely that there is no node at the precise coordinates the hash function produces. We define the *home node* for a GHT packet to be the node *geographically nearest* the destination coordinates of the packet. The home node serves as the rendezvous point for **Put()** and **Get()** operations on the same key.

Because a GHT packet is not addressed to a specific node, but rather only to a specific location, it is treated by GPSR as a packet bound for a disconnected destination: no receiver ever sees the packet addressed to its own identifier. We observe that GPSR will route such a packet to the appropriate home node. GHT uses GPSR's perimeter mode to find these home nodes. Under GHT, the packet enters perimeter mode at the home node, as no neighbor of the home node can be closer to the destination. The packet then traverses the entire perimeter that *encloses* the destination, before returning to the home node [13]. We name this perimeter the *home perimeter*. Under GHT, the home node knows to consume the packet when it returns after this tour of the home perimeter.

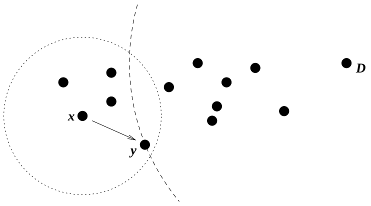

Figure 1: Greedy Forwarding Example: x forwards to y, its neighbor closest to D.

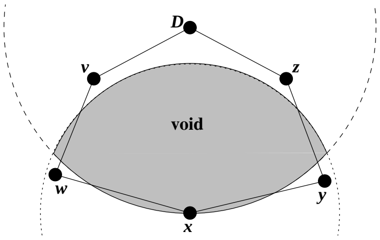

Figure 2: Void Example: x has no neighbor closer to D.

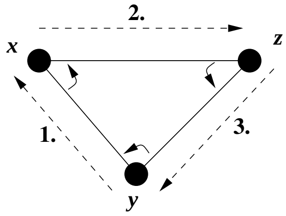

Figure 3: Right-hand Rule Example: Packets travel clockwise around the enclosed region.

With only the home node binding mechanism we've described thus far, GHT will work on static network topologies. Note that when the network topology changes after node failures, deployment of new nodes, or mobility, the identity of the home node and membership of the home perimeter may change. But for any snapshot of the network topology, there exist a home node and enclosing home perimeter for every location in the network. To offer persistence and consistency under the topological dynamics that sensornets are sure to exhibit, GHT needs a protocol to replicate key-value pairs, and re-associate them with the appropriate home node when the topology changes.

4.3 Perimeter Refresh Protocol

GHT uses the *perimeter refresh protocol* (PRP) to accomplish replication of key-value pairs and their consistent placement at the appropriate home nodes when the network topology changes. Recall that GHT routes all packets on a tour of the home perimeter that encloses a destination location. PRP stores a copy of a key-value pair at *each node on the home perimeter.*

PRP distinguishes between the home node and other nodes on the home perimeter, the *replica nodes.* A node becomes a home node for a particular key when the **Put()** packet arrives after completing its tour of the home perimeter. (This condition is detectable because GPSR writes the identity of the first edge a packet takes on a perimeter into the packet; the perimeter has been toured precisely when the packet arrives in perimeter mode and would be forwarded next on the same directed edge written in the packet as the first perimeter edge taken.)

PRP generates *refresh packets* periodically using a simple timer scheme. Every T_h seconds, the home node for a key generates a refresh packet addressed to the hashed location of that key. The refresh contains the data stored for that key, and is routed exactly as are **Get()** and **Put()** packets in GHT. Thus, the refresh packet will take a tour of the current home perimeter for that key, regardless of changes in the network topology since that key's insertion.

When a refresh packet arrives at a node, there are two possibilities: either the receiver is closer to the destination than the originator, in which case the receiver consumes the refresh packet and initiates its own; or the receiver is not, in which case it forwards the refresh packet in perimeter mode. In both cases, the receiver appends any additional key-value pairs it has stored for that key to the refresh packet. When a refresh packet returns to its originator, and that node was not previously the home node for that key, it consumes the refresh packet, and transitions to being the home node for that key. That is, the new home node sets its own refresh timer, and subsequently originates refreshes for that key. This mechanism provides the design goal of consistency: it ensures that the node closest to a key's hash location will become the home node for that key and store that key's data after topological changes.

When a replica node receives a refresh packet it didn't originate, it caches the data in the refresh, and sets a takeover timer for that key, T_t. This timer is reset every time a refresh for that key from another node arrives. Should the timer expire, the replica node initiates a refresh for that key and its data, addressed to the key's hashed location. The replica nodes and takeover timer provide persistence when nodes fail. When the home node for a key fails, its replica nodes will note the absence of refreshes for that key from its home node, and step forward to initiate refreshes. A replica node may or may not itself be the new home node; the GHT routing procedure causes the refresh to reach the new home node.

All nodes that hold data for a key, both home nodes and replica nodes, expire keys they cache when the death timer, T_d, expires. The death timer is reset every time a node receives a refresh message for that key, whether from itself or from another node. Clearly, $T_d > T_h$ and $T_t > T_h$. That is, a home node expires a key-value pair after failing to receive back multiple refreshes it originates, and a replica node waits for multiple home node refresh intervals to elapse before stepping forward to send a refresh for it. These choices of timer values make the PRP robust against episodic loss of its refresh packets. In the GHT system we evaluate herein, $T_d = 3T_h$, and $T_t = 2T_h$.

Figures 4 through 6 show an example of the operation of the PRP. Here, key k hashes to location L. After a **Put()** of (k, v), node a becomes the home node, and sends a refresh to L containing (k, v). Figure 4 shows the home perimeter enclosing L after this refresh has returned to a. Suppose that node a fails. After time T_t elapses, during which node d receives no refreshes from node a, node d sends a refresh to L containing (k, v), as shown in Figure 5. This refresh is delivered to node f, which becomes the new home node for (k, v). Figure 6 shows the network after f has sent a refresh that has returned to it, and the replicas it has recruited along the new home perimeter about L.

It is important to note that the PRP typically generates very local network traffic. On dense networks, perimeters are quite short (most perimeters in a dense network are three hops in length). When a home node moves, the refreshes it generates won't have far to travel before reaching the home perimeter, under reasonable assumptions of mobility rate and radio range (that is, that a node doesn't move many radio ranges in a period shorter than T_h).

The PRP also includes a *join optimization*, 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 more rapid re-establishment of a consistent home node when nodes fail or move.

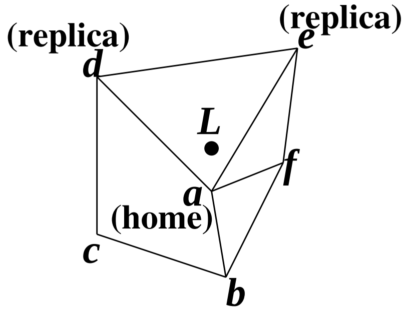

Figure 4: Key stored at location L, home node a, replicas d and e on the home perimeter.

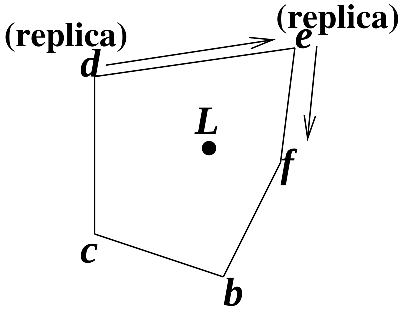

Figure 5: Time T_t after node a fails, replica d initiates a refresh for L.

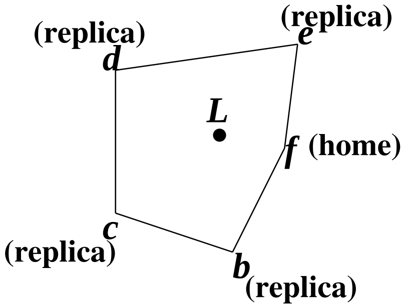

Figure 6: Node f becomes the new home node, and recruits replicas b, c, d, and e.

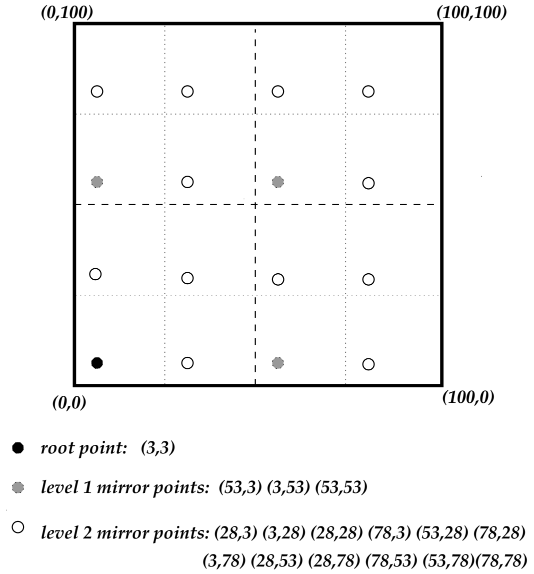

Figure 7: Example of Structured Replication with a 2-level decomposition.

4.4 Structured Replication

Thus far, GHT stores all events with the same key in the same place. If too many events with the same key are detected, that key's home node could become a hotspot, both for communication and storage. GHT employs *structured replication* (SR) to address this scaling problem. 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 [16]). Let us name the single location GHT hashes a key name into the *root* of that key. 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 GHT scheme without mirrors. For example, Figure 7 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, which is easily computable. Thus, SR reduces the storage cost at one node for one key with n detected events from $O(\sqrt{n})$ to $O(\sqrt{n}/2^d)$. GHT must now route queries to all mirror nodes, however. It does so recursively; first it routes a query to the root node, then from the root node to the three level-1 mirror points. Each of these in turn forwards the query to the three level-2 mirror points associated with them. This recursive process continues until all mirrors are reached. Responses traverse the same path as queries but in the reverse direction—up the hierarchy toward the root. Thus, a single query incurs a routing cost of $O(2^d\sqrt{n})$ as compared with $O(\sqrt{n})$ for GHT without mirrors. 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, SR reduces the cost of storage but increases the cost of queries. SR offers an intermediate solution between the local storage canonical method, where storage is free but queries expensive, and GHT without SR, where both are of moderate cost.[4] We expect that SR 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.[5]

5. SIMULATION RESULTS

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. After verifying the correct functioning of GHT and measuring its performance on static networks, we then consider the system's behavior in simulations with both failing nodes and mobile nodes, to test the system in the harsh sensornet environment.

After confirming the viability of our design, we then (Section 5.2) verify the scaling arguments from Section 2.3.2 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, node failures, or mobility.

5.1 Small-Scale Networks, Wireless Details

We implemented GHT in *ns-2* [17], which supports detailed simulation of mobile, wireless networks using IEEE 802.11 radios. In these simulations, we seek to demonstrate GHT's robustness 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 the 250-m radio range of IEEE-compliant hardware; this choice mirrors that made in the evaluation of directed diffusion in [10], in the

[4] Choosing d such that $2^d = \sqrt{n}$ costs the same as local storage.

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

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: GHT simulation parameters in *ns-2* simulations.

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 GHT on Static Networks. Results are the means of three simulations.

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 GHT. Stationary nodes, varied fraction of nodes alternate between up and down states. Results are the means of eight simulations.

interest of using parameters closer to those found in sensor radios. Our radio model is realistic in its use of the 802.11 MAC protocol for floor acquisition, and in its modeling of capture; these aspects reflect the contention behavior of today's commodity off-the-shelf radios. However, we do not consider environmental noise or propagation obstacles, and leave examination of their important effects to a future implementation study.

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.

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.[6]

In measuring GHT's performance, we are concerned with the *availability* of the data stored to queriers, and the *load* placed on nodes participating in GHT, 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 GHT: 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 GHT to hold data written to it.

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.

Note that we do not measure the routing protocol load placed on the network by GPSR in our simulations; we are evaluating GHT, not the underlying routing system, as is the practice in the evaluation of DHT systems [6, 21, 24, 25]. GPSR generates a constant volume of routing protocol traffic (beacons) per node, regardless of system size in nodes [14]; this load is of lower order than that generated by GHT, which sends packets on paths of length $O(\sqrt{n})$. Moreover, there is *no location database* like GLS [16] used with GHT, as GHT sends no traffic to node IDs.

5.1.1 Stable and Static Nodes

As one would expect, on static networks, where the topology doesn't change, GHT 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 GHT 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 GHT 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 neighbors. 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 their destina-

[6]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 [14].

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 GHT. Stationary nodes, all alternate between up and down states of varied lengths. Results are the means of four simulations.

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 GHT 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.

tion 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 GHT 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, GHT's ability 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 GHT when the join optimization and refreshes fail to keep events alive in GHT.

5.1.3 Stable but Mobile Nodes

Table 5 shows how GHT performs on a mobile sensor network of 100 nodes. In these simulations, nodes move using the random waypoint model [17]; 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.

Under node mobility, GHT 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 GHT 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 [14]. 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 GHT, 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

As expected, GHT works well in sensornets with stable and static nodes. But failures (and movement, in some cases) are inevitable, 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 uses 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 isn't overly high, or the cycle times are tens of minutes, the system performs well. Similarly, the extent and rate of movement in the mobile node case is significant; nodes rest only a minute between movements, and the movements are large excursions (half the size of the sensornet, on average), not slight adjustments. Here, too, availability remains high.

Our GHT algorithm replicates a key-value pair at nodes in the immediate vicinity of the home node. Localized replication of this form is of little use if all the 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 at dispersed locations (using multiple hash functions).

5.2 Comparative Study

The detailed *ns-2* simulations verify the correctness and robustness of the GHT system in a realistic wireless environment, including MAC-layer behavior, packet loss, node dynamics, *&c.* However, they were limited to system sizes on the order of 100 nodes. We now use less detailed simulations to compare the three canonical mechanisms—external storage (ES), local storage (LS), and data-centric storage (DCS)—in much larger systems. We built a special-purpose simulator that assumes that nodes are stable and stationary and that packet delivery to neighboring nodes is instantaneous and error-free. This simulator thus faithfully represents the packet generation and forwarding behavior 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 sensornet and workload vary, to illuminate the *relative* performance of the canonical data dissemination algorithms. We do not count PRP packets in these simulations. The length of a perimeter is purely determined by the density of the network, and we only vary system scale in nodes, not density, in the large-scale simulations.

We use two metrics to evaluate the performance: the total number of packets generated, and the hotspot usage, the maximum number of packets transmitted by any single node. 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

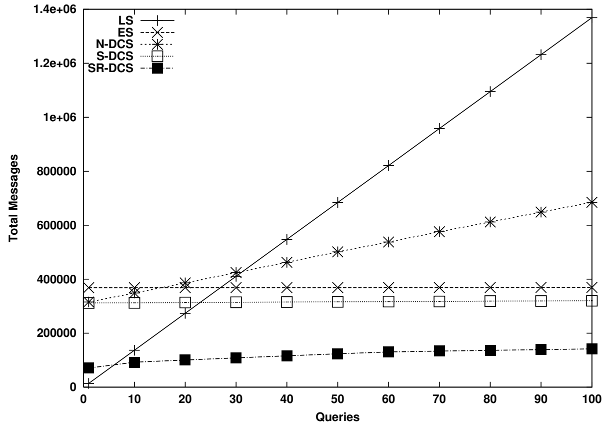

Figure 8: Total number of messages generated as Q, the number of event types queried for, is increased. The number of nodes (n) is held fixed at 10,000 nodes.

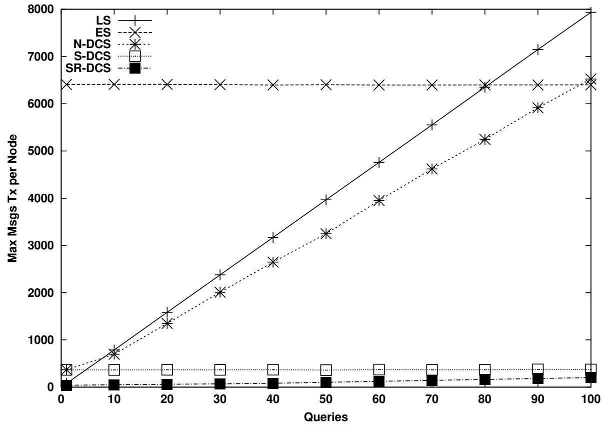

Figure 9: The maximum number of messages sent by any single node as Q, the number of event types queried for, is increased. The number of nodes (n) is held fixed at 10,000 nodes.

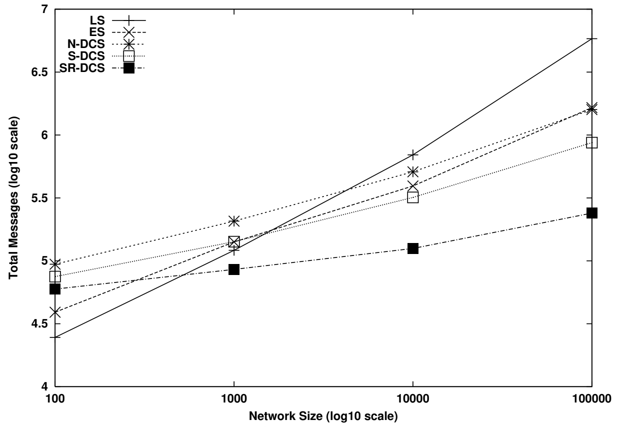

Figure 10: Total number of messages generated as n, the number of nodes, is increased. The number of event types queried for (Q) is held fixed at 50.

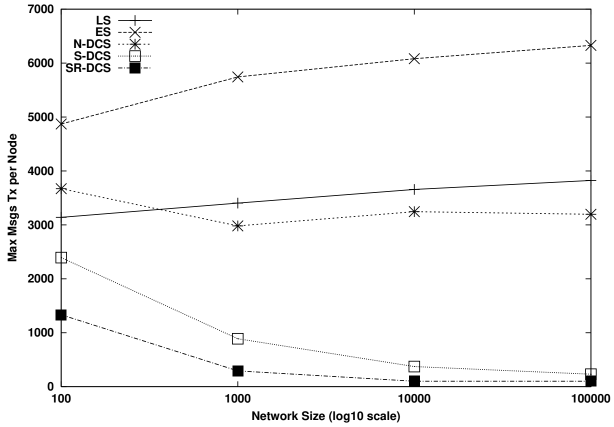

Figure 11: The maximum number of messages sent by any single node as n, the number of nodes, is increased. The number of event types queried for (Q) is held fixed at 50.

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 DCS:

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

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

Structured Replication DCS (SR-DCS): We assume an optimal level of SR (as described in Section 4.4) to provide a lower bound. We assume summarization in this case.[7]

5.2.1 Test #1: Varying Q

The results from varying Q are shown in Figures 8 and 9. For low Q, LS has low total and hotspot usage, but both quantities increase linearly in 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 DCS that use summarization have low total and hotspot usage, but note that structured replication in SR-DCS reduces the total usage significantly. The hotspot and total usage of DCS without summarization (N-DCS) increases linearly in 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-DCS being the best. For high Q, SR-DCS is the clearly superior choice, followed by S-DCS. If summarization is not allowed, then the choice is between the lower hotspot usage of N-DCS and the lower total usage of ES.

5.2.2 Test #2: Varying n

The results from varying n appear in Figures 10 and 11. 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-DCS and SR-DCS have the lowest hotspot usage by far, but among methods without summarization DCS and LS have similar performance. ES has the worst hotspot load. Thus, at all but the lowest values of n (lower than around $n = 1000$) the DCS variants are the superior choices. Recall that these simulations use $Q = 50$, 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 2.3.2; the only significant deviations arise in cases where the hotspot usage does not occur at the access point. These simulations, while idealizing wireless link behavior, were true packet-level simulations of these algorithms in systems as large as $n = 100,000$. The formulae of Section 2.3.2 suggest that DCS is particularly appropriate as system size grows

[7]We omit structured replication without summarization in the interest of brevity.

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 events are summarized in responses).

6. RELATED WORK

Directed diffusion [9, 10] is an example of *data-centric routing*—routing based on the name of the data rather than on the identity of the destination node. Unlike our proposed DCS mechanism, event information in directed diffusion is stored locally at the detecting node; as such, directed diffusion is closer to the local storage (LS) model. Directed diffusion also provides additional mechanisms for the *reinforcement* of high-quality data delivery paths and for *in-network aggregation* (*i.e.,* as the data is being routed to the requestor, it may be aggregated by intermediate nodes). Directed diffusion doesn't require geographic information; it uses flooding.

The Geographic Location System (GLS) used in GRID [16] can be augmented to provide a DCS-like service. Geographic routing delivers packets to locations, not addresses; thus, a packet sender must be able to map a destination's identifier to its geographic location. GLS is a scalable location service that performs this mapping. 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 geographic location. GLS 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 event names 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 GLS provides this location database itself as a service, GHT avoids this level of indirection and instead maps event names directly to locations.

The SCOUT [15] location tracking system might also be used similarly. While SCOUT uses hierarchical addressing and routing based on landmark routing, GHT uses GPSR, a flat routing algorithm wherein nodes are addressed with geographic coordinates.

Although GHT was inspired by Distributed Hash Table systems like Chord and CAN [6, 21, 24, 25], we did not adopt the routing algorithms used in these systems. These algorithms require nodes to be interconnected in a fairly rigid manner. On the Internet, node neighbor relationships are at the *logical* level; the underlying IP routing system logically connects nodes that are not immediate physical neighbors. It is not clear how (if at all) the node connectivity required by these DHT algorithms can be efficiently achieved in a sensornet environment. GPSR allows us to achieve the required hash-table functionality while working with only the true physical connectivity between nodes.

7. CONCLUSION AND FUTURE WORK

This paper presented the design and evaluation of GHT, a DCS system for sensornets built on geographic routing. We have predicted analytically and verified in simulations of networks of up to 100,000 nodes the cases where DCS offers reduced total network load and hotspot network usage as compared with external storage and local storage. Our analysis reveals that these benefits occur on sensornets comprised of large node populations, and where many events are detected, but not all event types are queried. GHT leverages the GPSR geographic routing system to offer an efficient DCS service that maintains persistence of data when nodes fail or move, while scalably spreading the load of (key, value) pairs evenly throughout the sensornet.

Several avenues beg further investigation. Foremost among these is the effect of varying node density. Under GHT, keys are uniformly hashed over the geographic space. Hence, as nodes are distributed increasingly non-uniformly, we expect the storage and forwarding load across nodes to be correspondingly skewed. We are investigating the consequent performance implications and developing mechanisms that can adapt to such non-uniformity.

To avoid hashing keys to points outside the sensornet, GHT requires approximate knowledge of a sensornet's boundaries.[8] Our work herein assumes foreknowledge of these boundaries (recorded, *e.g.,* when the network was first deployed). An open research problem is to devise scalable distributed algorithms to map these (possibly changing!) geographic boundaries.

Finally, our proposed design fundamentally requires that a node know its own geographic position. While this assumption seems reasonable for most sensornets, an open question is how one might achieve DCS using only approximate geographic information, or without any such information.

8. 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*, Charleston, SC, USA, Dec. 1999, pp. 186–201.

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

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

[4] 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, Apr. 2001.

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

[6] P. Druschel and A. Rowstron, 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)*, Nov. 2001.

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

[8] 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.

[9] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, Building efficient wireless sensor networks with low-level naming, In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Banff, Alberta, Canada, Oct. 2001.

[10] 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)*, Boston, MA, USA, Aug. 2000.

[11] J. M. Kahn, R. H. Katz and K. S. J. Pister. Mobile networking for smart dust, In *ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*, Seattle, WA, USA, Aug. 1999.

[12] B. Karp, Greedy perimeter state routing, invited seminar at USC/ISI, Arlington, VA, USA, Jul. 1998.

[13] B. Karp and H.T. Kung. GPSR: greedy perimeter stateless routing for wireless networks, In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, Boston, MA, USA, Aug. 2000.

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

[15] 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, Nov. 2000.

[16] J. Li, J. Jannotti, D. DeCouto, D. Karger, and R. Morris, 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.

[17] S. McCanne and S. Floyd. *ns* network simulator, `http://www.isi.edu/nsnam/ns/`.

[18] G. Pottie and W. Kaiser, Wireless sensor networks, *Communications of the ACM*, 2000.

[19] 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)*, Boston, MA, USA, Aug. 2000.

[20] 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)*, Rome, Italy, Jul. 2001.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, A scalable content-addressable network, In *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001, pp. 161–172.

[22] 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)*, Rome, Italy, Jul. 2001.

[23] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, Data-centric storage in sensornets, Under submission to *ACM SIGCOMM HotNets*, Jul. 2002.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications, In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, CA, USA, Aug. 2001.

[25] B. Y. Zhao, J. Kubiatowicz, and A. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, EECS Department, 2001.

[8]When a **Get()** or **Put()** hashes to a point outside the external perimeter, the packet will walk the entire external perimeter before arriving at the home node. In such cases, GHT operates *correctly* albeit less efficiently.