Title

Using Hierarchical Location Identifiers for Scalable Routing and Rendezvous in Wireless Sensor Networks

Authors

Bian, Fang
Li, Xin
Govindan, Ramesh
et al.

Publication Date

2005-05-05

Peer reviewed

Using Hierarchical Location Names for Scalable Routing and Rendezvous in Wireless Sensor Networks

Fang Bian* Xin Li* Ramesh Govindan* Scott Schenker†

Abstract

Until practical ad-hoc localization systems are developed, early deployments of wireless sensor networks will manually configure location information in network nodes in order to assign spatial context to sensor readings. In this paper, we argue that such deployments will use hierarchical location names (for example, a node in a habitat monitoring network might be said to be node number N in cluster C of region R), rather than positions in a two- or three-dimensional coordinate system. We show that these hierarchical location names can be used to design a scalable routing system called HLR. HLR provides a variety of primitives including unicast, scoped anycast and broadcast, as well as various forms of scalable rendezvous. These primitives can be used to implement most data-centric routing and storage schemes proposed in the literature; these schemes currently need precise position information and geographic routing in order to scale well. We evaluate HLR using simulations as well as an implementation on the Mica-2 motes.

1 Introduction

Data-centric abstractions for routing and storage have received a fair amount of attention in the research literature. Data-centric routing systems such as Diffusion [7] and TinyDB [12] have been used in many sensor network deployments. A body of literature has proposed a complementary class of data-centric storage systems [23, 21, 11] that supports the construction of distributed hash tables and indices for scalable querying.

While many advances have been made in designing data-centric abstractions, not much attention has been paid to the underlying packet routing and rendezvous primitives. Existing implementations of data-centric abstractions use two kinds of packet routing primitives: flooding and geographic routing. Anecdotal evidence from current deployments suggests that flooding adversely impacts the performance even in networks with tens of nodes. The alternative, geographic routing using protocols like GPSR [8], requires assigning position information to nodes. Such information is generally expected to be dynamically computed using an ad-hoc localization system [1, 5, 10], or a system that assigns virtual coordinates [19, 15] for routing purposes. These systems are currently the subject of active research, and practical deployments are perhaps a few years away.

This paper considers an alternative approach to providing routing primitives for data-centric abstractions. Our approach is based on the observation that without dynamically computed node *positions*, most near- to medium-term sensor network deployments will configure node *locations*.[1] Node location provides context for the data collected from the sensor network. Such location information is often loosely associated with geography or topography. Thus, in a habitat monitoring network, a node might be located within the "chaparral" region or within a "riparian" region. In an in-building network, the location of a node may be specified by floor and wing (*e.g.*, 13th floor, west wing). Furthermore, location names often have a natural *hierarchy*. In a habitat monitoring network, such a hierarchy might be defined by, for example, a quadrant of the habitat, followed by a section, and within it a particular cluster of nodes. In a building network, the hierarchy might be defined by floors, wings and rooms. A hierarchical location naming scheme is more user-friendly than a system in which nodes are manually assigned positions. In fact, we know of at least two deployments that use such a naming scheme to assign spatial context to sensor readings.

In this paper, we consider deployments where nodes are configured[2] with *hierarchical location identifiers* (HLIs). An HLI is simply a machine readable form of a hierarchical location name. Thus, a sensor node in a building might be assigned an HLI of the form 5.4.10 where 5 denotes the fifth floor, 4 denotes the east wing, and 10 denotes the

*Computer Science Department, University of Southern California, Los Angeles, CA 90089, USA. Email: {bian, xinli, ramesh}@usc.edu

†Computer Science Department, University of California, Berkeley, Berkeley, CA 94720, USA. Email: shenker@icsi.berkeley.edu

[1] We use position to denote the precise position of a node in some geometric coordinate system, and location when other forms of expressing where a node is situated are used.

[2] In Section 2, we discuss mechanisms for configuring these nodes that do not require significant manual intervention.

10th room on the east wing.

The central thesis of this paper is that these HLIs can be used to build a scalable routing system (which we call HLR) for sensor networks. Observe that the HLI hierarchy can be modeled as an *area* hierarchy [9]. Imposing an area hierarchy on a network is a well-studied way of scaling routing protocols in wired and wireless networks. In HLR, the location naming hierarchy implicitly defines an area hierarchy; by contrast, in wired networks, other factors such as organizational boundaries or cabling costs might determine the design of area hierarchies. Thus, in our example above, a node 5.4.10 is in the 5-th top-level area, the 4-th second level area within the top-level area and so on. Nodes in an area hierarchy maintain detailed routing information about nodes within their area, and less detail about nodes outside their area. In HLR, for example, the node 5.4.10 would have a routing table entry for all nodes within the area 5.4, one entry for each of the sub-areas of 5, and one entry for each of the top-level areas.

HLR constructs and maintains these routing tables using a variant of the distance-vector based routing protocol DSDV [17]. While the basic design of HLR borrows heavily from the routing literature for wired networks, it incorporates two novel features. The first is a technique for automatically *aggregating* routing entries at area boundaries that allows neighboring areas to maintain summarized views of an area. The second is a mechanism for *routing to partitioned areas*—classical area hierarchy based algorithms make the assumption that areas are connected.

Using the routing tables that HLR constructs, it is possible to provide a variety of packet routing primitives: *unicast* to a specified node within the network, *broadcast* or *anycast* to a specified area, *rendezvous* using a *random* hash or a *locality-preserving* mapping. Particularly novel in HLR is the design of the rendezvous primitives, since previous designs of such primitives for sensor networks leveraged geographic positioning. These primitives can be used for data-centric routing systems like Diffusion and TinyDB, as well as for data-centric storage systems like GHT [21] and DIM [11].

We have implemented HLR in TinyOS, and have implemented simplified versions of data-centric routing and storage systems that use HLR's routing primitives. We use extensive simulations to compare the performance of HLR-based data-centric routing and storage to systems that use geographic routing. We find that the performance of the two classes of systems is comparable; while aggregated route entries increase the average path length in HLR, geographic routing based rendezvous sometimes incurs significant overhead in walking the outer perimeter. We also evaluate the behavior of HLR under dynamics, finding that route changes caused by link failures can often be constrained to a small area and are not propagated throughout the network. Finally, we report experiences from running HLR on a small-sized network of Mica2 motes. Taken together, these results imply that HLR is a viable routing layer for many kinds of sensor networks that can be immediately employed in near-term sensor network deployments.

Our reliance on configured node addresses may seem to be awkward, given the networking community's experience with manual configuration in the Internet context. We make two observations in our defense. First, many Internet components (the backbone routing system, the name system) are still manually configured. Second, unlike the Internet which is comprised of different administrative organizations, sensor networks are likely to be managed and deployed by one organization. Furthermore, until precise self-localization technology is deployed, we expect that sensor network deployments will need to be carefully planned, with human involvement in identifying each node's position. Given this, it is a small step to configure these positions on nodes (and techniques can be developed to reduce the error in this configuration step).

2 Overview and Related Work

In this section, we discuss the feasibility of Hierarchical Location Identifiers(HLI) for sensor network deployments. Then, we briefly list the routing primitives HLR provides and how we make use of HLI to build HLR. Finally, we compare HLR to the other related work.

2.1 Feasibility Discussion

A fundamental premise behind our approach is that most sensor network deployments will need to associate nodes with names in order to make sense of generated data. Often, these names will have location information embedded in them. An example of such a name is: (Residence Hall 1).(Third Floor).(West Wing).(Sensor 5). Such names have two natural properties that reflect the way humans think about sensory data acquisition: they are *hierarchical*, and they contain some location information embedded in them that is usually imprecise (*i.e.,* not a position in some coordinate system). We term the machine readable numeric ID translated from these hierarchical string names as Hierarchical Location Identifiers(HLI).

In this paper, we observe that these hierarchical identifiers (HLIs) indicate approximate topological proximity of the nodes and can therefore be leveraged to build scalable routing primitives for wireless sensor networks. Before we discuss how to make use of HLI to support scalable routing primitives, we first discuss mechanisms for node HLI's assignment.

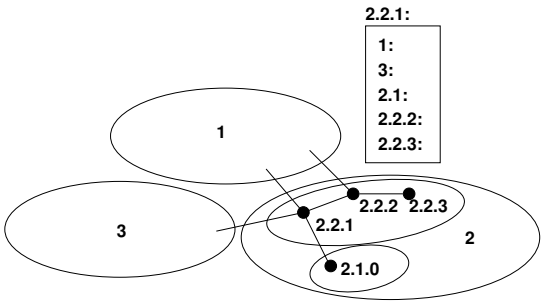

Figure 1: Example: a sensor network with HLI and routing table of node 2.2.1 built by HLR

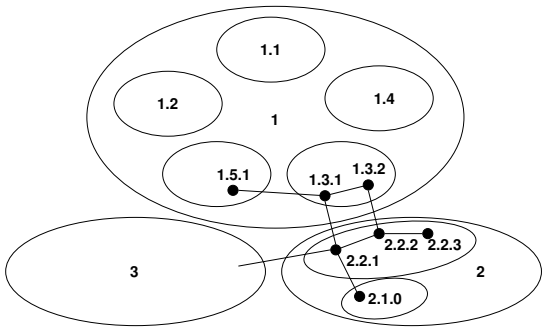

Figure 2: The same sensor network in Figure 1 with more details in area 1 shown.

Consider a sensor network deployment in a building or a habitat. We expect that most such deployments will be *planned*: a domain expert will need to determine where to place the sensors, how many to place, *etc.* When planning this deployment, the network administrator needs some way to associate the data received from a sensor node with its location. In the absence of localization, the administrator will likely have to manually create a "database" that maps node identifiers to human-readable location identifiers. These location identifiers are often hierarchical, and we argue that it is feasible to use this database to configure HLIs for nodes.

This HLI configuration can be automated in several ways.

For example, the administrator can "zap" each individual sensor device with its HLI before deployment. Alternatively, one can design a simple bootstrap protocol (similar to DHCP) by which a node obtains its HLI. The design of such a protocol is well-understood but is a bit beyond the scope of this paper. In this way, HLR simply leverages the fact that most sensor network deployments will be planned, and does not add any additional human involvement beyond what will be required for such deployments anyway.

2.2 Overview

Equipped with HLIs, our work shows that scalable routing could be designed for wireless sensor networks. The key insight behind routing using HLIs (HLR) is that one can use HLIs automatically and dynamically build an aggregated routing table that scales well with network size. Behind this insight lies the assumption that hierarchical location naming is also approximately topologically-congruent to node placement. That is, all nodes whose HLIs begin with 1 are situated within some well-defined geographic region. In our example, all such nodes would be within Residence Hall 1. This property also applies recursively, so that all nodes whose HLIs start with 1.3 are on the 3rd floor of Residence Hall 1. In situations where this is the case, HLR can build compact and accurate routing tables. For example, in the sensor network shown in Figure 1, by running HLR, node 2.2.1's routing table will contain one entry to the whole area 1, one entry to area 3 and one entry to its sibling area 2.1 in addition to one entry for the other nodes in the same area like 2.2.2 and 2.2.3. So, assuming the hierarchy of the network is appropriately designed, the size of the routing table can grow *logarithmically* with network size.

Scalability is not the only advantage of HLR. In addition to supporting unicast, HLR could also be used to support area-based multicast or anycast. Unicast can be used for tasking individual nodes. Area-based multicast enables any node in network to deliver a message to a subset of nodes which shares a common prefix in their HLIs, For example, area-based multicast can be used in TinyDB to deliver a query to a set of nodes around a monitored plant to start collecting data. This primitive cannot be easily supported by any geographical location-based routing using location information.

However, that is not the only advantage of HLR. In this paper, we also show how easily we can leverage HLR to support rendezvous-based primitives such as *hash-lookup* and *data-locality preserving hashing*, which are important building blocks for data-storage systems proposed for sensor networks. Hash lookup can be used to implement functionality equivalent to a single key data storage system such as the GHT [21], while the last primitive can be used to build a locality-preserving data storage system which supports multi-dimensional queries such as the DIM [11].

To our knowledge, these are *all* the primitives that have been proposed for use for data dissemination and querying in sensor networks. That HLR can support them without requiring ad-hoc localization systems is its main selling point.

The challenges in the design of many of these primitives, and in the design of HLR itself, lie in dealing with

dynamics like route changes and partitioned areas. We discuss in detail the design of basic HLR in Section 3 and the design of routing and rendezvous-based primitives in Section 4.

2.3 Related Work

In this section, we discuss related work that have inspired the design of HLR and contrast our work to the other sensor network routing proposals.

Hierarchical routing has been a subject of research for decades [9, 26, 22]. Today's Internet, for example, is built on top of hierarchical routing schemes such as BGP [22] and OSPF [14]. A hierarchical routing scheme such as ours will provide the scalability needed by these large scale sensor networks.

The ad hoc networks community has also been working on hierarchical network organizations [18, 16] where the primary goal is to provide a reliable communication infrastructure for node mobility management. For a large class of sensor network applications, mobility is not an issue, and HLR does not attempt to solve the dynamics resulting from node mobility.

Clustering schemes [26, 6, 4] represent a related part of the literature. In general, such schemes (particularly as proposed for sensor networks) are somewhat orthogonal to HLR, since they are explicitly focused on node energy management. HLR clusters nodes into areas for routing information scaling.

Geographic routing schemes [8] are complementary to HLR. They rely on accurate position information, while HLR relies on logical location names. Automatically determining position information (localization) is still the subject of much research. Of course, a sensor network deployment could use configured position information, but this might require significant manual labor, especially in environments where GPS signals might not be readily available.

Recently, several virtual coordinate schemes have been proposed to support stateless location-based routing [19, 15]. It is unclear that these schemes can be used for routing without the development of another service that maps a node's identifier to a virtual coordinate (since a virtual coordinate has almost no relation to the physical coordinate). Such schemes cannot be used for data-centric storage as well without incurring significant data migration overhead when virtual coordinates change.

3 HLR Details

In this section, we discuss the details of HLR. We start by discussing an overview of HLR performance, then describe its aggregation and robustness mechanisms in some detail.

3.1 Overview

HLR assumes that HLIs have been configured into network nodes. As discussed above, the fundamental premise our paper makes is that deployments will, in the absence of localization, need to maintain a mapping between some node identifier (perhaps drawn from a flat name-space) and some textual description of a location. Typically, this information will be maintained in a database or file. HLR only requires that, in addition to (or perhaps instead of) the textual description of location, a network administrator assign a hierarchical location identifier (or HLI) to each node. Thus, for example, the administrator needs to translate a location identifier like "node 1 in floor 5 of building 1" into an identifier of the form 1.5.1.

When a node is assigned a HLI such as 1.5.1, we say that it belongs to the *top-level area* numbered 1, the second-level area 5 of the top-level area 1, and so on. We say that the top-level area has a *depth* of 3. In HLR, different top-level areas are allowed to have different HLI depths, *e.g.*, 1.5.1 and 2.1.

HLR is fairly minimal in its assumptions about what nodes need to be configured with. It only requires that each node know its own HLI. Thus, a network can be incrementally deployed without having to reconfigure existing nodes.

The key insight behind HLR is that one can automatically and dynamically construct *aggregated routing tables* with the configured HLI at each node, using a modified version of a distance-vector algorithm such as DSDV [17]. DSDV is a distance vector routing algorithm which associates a sequence number with each destination to avoid the "count to infinity" problems associated with distance-vector protocols. In the basic DSDV, each node advertises a route to itself, and associates that route with a monotonically increasing sequence number. Neighboring nodes periodically exchange distance vectors to each destination, together with the sequence number of each destination. To a given destination, a node might possess several routes heard from each of its neighbors. Of these routes, a node only considers routes assigned the most recent sequence number. There may be more than one such route corresponding to different paths to the destination, but the key intuition is that all routes with the same sequence number represent a *consistent* view of paths to a destination. From these routes, each node picks the shortest, and advertises that to its neighbors. This intuition also explains why DSDV avoids the count-to-infinity problem associated with earlier distance vector algorithms; at any instant, the routes selected by each node to a destination taken together form a tree rooted at the destination node, which

by definition is acyclic.

The main challenge in adapting DSDV to HLR is *route aggregation*. The goal of HLR is to scale the routing table such that, for example, in a network as shown in Figure 2, the node with HLI 1.3.1 should have:

- One route to each node in area 1.3, such as 1.3.2, 1.3.3, and so on.
- One route to each of the sibling areas of 1.3, such as 1.1, 1.2, 1.4, 1.5 and so on. A route to, say, area 1.5 is said to be an *aggregated route*. Aggregation is the fundamental contributor to scaling the Internet as well as HLR.
- One aggregated route to each top level area other than its own, such as 2, 3, and so on.

Depending upon how the HLIs are assigned, such a routing table can scale logarithmically with network size.

In HLR, we accomplish route aggregation automatically using a simple modification to DSDV. The intuition for doing this comes from the following observation. Consider the network shown in Figure 2, suppose that nodes 1.3.1 and 1.5.1 are neighbors of each other. Then, the former can create a route for the 1.5 *aggregate*, when it hears a route advertisement from 1.5.1. Thus, any packets destined towards *any* node in 1.5 from 1.3.1 will be forwarded to 1.5.1. This aggregation relies on an important property: all nodes within 1.5 (and more generally, any area) are *connected* (*i.e.*, there exists a path between two nodes in an area that does not exit the area). For now, we assume that this connectivity assumption is satisfied. Later in this section, we will discuss how HLR can be adapted to deal with situations when an area is internally partitioned.

For context, most of what we have discussed above is well-known in the routing literature; area hierarchies have been studied for a long time. However, our contribution here is the design and implementation of a distance vector protocol for wireless sensor networks that performs *automatic route aggregation*. In wired networks, link-state protocols like OSPF perform these kinds of aggregation, but we do not know of actual designs or prototypes of distance vector protocols that have been augmented to automatically aggregate routes.

How does HLR perform this aggregation?
As we discussed above, instead of maintaining routes to individual nodes, HLR conceptually maintains routes to *areas*. At the boundary of an area (such as the one between 1.3.1 and 1.5.1 in our example above), nodes aggregate routes to areas. A node can detect that one of its links intersects an area boundary by comparing its own HLI with that in the route it hears. A node may hear many routes to an area, potentially one from each "gateway" node (a node which has at least one link that intersects the area boundary); if so, it picks one of these, and propagates it to its neighbors.

Unlike DSDV which conceptually builds a tree rooted at the destination, HLR builds, for each destination area, a forest with trees rooted at the area's "gateway" nodes. In this way, it maintains DSDV's loop-freedom and reduces the number of routing packets since each node only needs to join and propagate one tree for each hierarchical destination area. For example, in the network shown in Figure 2, all nodes in area 2 only need to keep track of one path to area 1, but not necessary the same path. For instance, node 2.2.1 and nodes in area 2.1 join the tree rooted at node 1.3.1, while node 2.2.2 and 2.2.3 use the route to node 1.3.2 as its path to area 1. Also, all nodes in area 2 only propagate one path to area 1 to area 3. Nodes in subarea 1.3 of area 1 need to keep track of path to the subareas 1.1 and 1.2 and so on.

What are the trade-offs in using HLR over vanilla DSDV for wireless sensor networks? Clearly, maintaining routes to every node in a wireless sensor network is neither feasible nor necessary. Yet, we argue that a protocol like HLR can be very useful in wireless sensor networks a) because its areas mirror logical location-based distinctions which often form the basis of user queries or network tasking instructions (*e.g.*, in an in-building network, many queries are likely to be expressed in terms of floors and wings), and b) HLR efficiently maintains routes to these. As we show later, HLR can be used to efficiently implement a variety of routing primitives in a highly scalable fashion, so the intuition here is that by expending a little energy to provide a general routing substrate, we can make the rest of the system significantly more efficient.

When used this way, HLR has one advantage and one disadvantage. Like any protocol based on area hierarchies, HLR does not provide optimal paths. However, as we will show in section 5, the performance of HLR is often comparable to, or better than other alternatives since those alternative often have other pathologies (*e.g.*, traversing the outer perimeter in a GHT). The advantage of HLR, and an important one from the perspective of sensor networks, is that most node or link failures only affect a small number of nodes (usually those within the failed node's own lowest-level area). We validate this in our simulations.

We have implemented HLR on Berkeley motes. Details of our implementation and some results from a small deployment will be discussed in section 6.

3.2 Automatic Route Aggregation

We now discuss, in some detail, the route selection, aggregation, and route propagation rules in HLR. For simplicity, in this discussion we assume that all areas are internally connected. In the next subsection, we discuss how

HLR relaxes this assumption.

In HLR, each node periodically exchanges *routes*. Each route is associated with the HLI of a destination *node*. This is an important point; HLR does not *propagate* routes to an area, and routes always refer to a node within an area. HLR does, however, *compute* and *store* routes to an area. When a node receives multiple routes to nodes within the same area, it picks one of those and re-advertises it. For example, consider a node 2.2.1 which receives five routes, one each to 1.5.1, 1.2.3, 1.3.1, 1.4.1 and 1.1.2. From the perspective of this node, *all of these routes represent paths to destinations in area* 1. We call area 1 the *effective destination area* from the perspective of node 2.2.1. Then, node 2.2.1 picks one of these routes, say the route to 1.3.1, and advertises that.

This is a subtle point; one would have expected HLR to be designed such that 2.2.1 would advertise the aggregate 1 instead of the route 1.3.1.[3] Doing so, however, without violating the semantics associated with the sequence numbers turned out to be tricky. This behavior of HLR defines the intuition described above: HLR maintains a forest of trees for a given area, and different nodes "join" different trees in this forest by picking the best available route. However, this choice has an interesting trade-off. If it had been possible to advertise the aggregate, then even if any one of the five selected routes had changed, that change would be hidden from nodes downstream of 2.2.1. Now, however, if the selected route 1.3.1 fails, another route will have to be selected and propagated[4], so this choice has weaker failure containment properties. In practice, though, as our simulations show, the performance of HLR is still quite good, and most failures affect only a small number of nodes.

Thus, each route is associated with an HLI of a node, a sequence number, a path metric to the destination node, and a lifetime associated with the route. The route lifetime is used to purge stale routes, and the sequence number for loop avoidance. In our simulations, we use the hop distance as the path metric. While this is known to be a bad choice in selecting paths in wireless networks [27], we augment this with link blacklisting (see below) in our current implementation. Longer term, we see using other additive path metrics that capture notions of link and path quality [27, 2] in HLR. HLR can be easily modified to include more sophisticated path metrics.

We now more precisely describe the route selection and aggregation rules. From our discussion above, this is the step in which the route aggregation is *implicitly* performed, since HLR does not propagate aggregates. Suppose that node A has n different routes that it has heard from its neighbors. It first partitions the set of routes such that all routes in a subset share an HLI prefix h defined as follows: if h has l elements, then the first $l-1$ elements of h and of A's HLI must be the same. The intuition, of course, is that h defines a distinct area outside A for which A need only maintain one route. Each subset also defines one *effective destination area*. In our example above, the 5 routes that node 2.2.1 has defines a subset. In this case, h is 1 and l is 1.

Now, consider a single subset. The node A selects exactly one route from this subset using the following rule. It further refines the subset by associating all routes to the same HLI into one cluster. From each cluster, it picks the lowest cost route with the most recent sequence number. Then, from within these selected routes, it picks the lowest cost route. These rules are basically designed to select the nearest "gateway" for the area corresponding to that subset. Different nodes select different gateways to a given area, and the chosen routes form a forest (as we have described earlier).

Having selected one route to each subset (or effective destination area), node A advertises these routes to its neighbors. In this manner, HLR scales well, since it maintains the property of hierarchical routing protocols: more detailed routing information about nearby nodes, and less about nodes farther away.

3.3 Dealing with Route Changes

HLR deals with route dynamics (addition of a node, failure of a link *etc.*) in ways similar to other routing protocols. Each route is associated with a *lifetime*, and must be refreshed at least once within that lifetime otherwise it is considered to have failed. HLR uses two frequencies of route advertisement. For a route that has recently changed, nodes re-advertise their routing tables with moderate frequency to allow for faster convergence. For routes that have been relatively stable, the route advertisement interval is set to be an order of magnitude higher. The lifetime is set to four times this longer interval. All the parameters are configurable in HLR.

Wireless links are known to be notoriously unstable, so dropped route advertisements are more likely to be the norm than the exception. Clearly, this can impact route stability: lost advertisements might result in route expirations. To avoid this, our implementation uses a simple link-layer black-listing scheme that filters out asymmetric links as well as highly lossy links, and paths are selected on the rest of the topology. When a link degrades and is marked unusable, the attached node performs the appropriate actions.

[3] Note that while 2.2.1 has 5 routes to area 1, it only re-advertises one of them, thus maintaining the desired scaling behavior.

[4] Unless all the routes to area 1 fail, this change will not trigger an instant propagation; rather, it will be propagated in the next regular advertisement.

3.4 Relaxing the Connectivity Assumption

In our discussions so far, we have relied on an important property, that of the connected-ness of an area. In practice, one would expect this condition to be *mostly*, but not always, satisfied. For example, in a building network, it might be reasonable to deploy sensors such that sensors within a floor are connected (using our definition in subsection 3.1). However, given the vagaries of wireless communication, it would be unwise to *rely* on this property for the correctness of the system. In this section, we show that we can add a little machinery to HLR's basic mechanism in order to deal with *partitioned* areas (where the connected-ness assumption is violated). Note that in our discussions below, we assume that while an area may be partitioned, the entire network is connected; HLR finds an alternate path to the sub-areas.

Our basic approach is to identify the partitioned areas by assigning a unique identifier (termed as cluster ID) to each connected component of the partitioned area. Nodes external to the area then "join" two different trees, one for each component: to them, different components look like different areas. However, data packets destined to a given HLI in the area are duplicated and sent to *both* partitions, since it is *a priori* unclear which partition contains the node associated with the HLI.

We now describe several details of this scheme. The first detail is the definition of a cluster ID; in HLR, nodes within an area settle on the lexicographically smallest HLI of any node within an area. For example, in the sensor network shown in Figure 2, the cluster ID of area 2 is 2.1.0. Notice that a sub-area of area 2 might have an entirely different cluster ID: thus, in our example, 2.2 would choose 2.2.1 as its cluster ID. Thus, if an area is partitioned into two, the two partitions will end up choosing different cluster IDs. We discuss below how this affects route selection. However, note that a basic property of HLR is that an area's partition is not visible outside the enclosing area as long as the latter itself is connected. In our example, assume area 2.2 is partitioned into two parts: one with cluster id 2.2.1, the other with cluster id 2.2.2. As long as area 2 is still connected, nodes 2.2.1 and 2.2.2 will see same cluster ID for area 2, which is 2.1.0. And thus the truth that area 2.2 is partitioned nodes is transparent to nodes in area 1 and area 3.

How do all the nodes within an area determine their cluster ID? In HLR, a node whose HLI is of the form *a.b.c* maintains one route to all top-level siblings of area *a*, all children of *a* who are siblings of *a.b*, and all nodes within *a.b*. Thus, for each *level*, just from its routing table, a node can determine the cluster ID. If there exists a partition at a particular level, then the connected components settle on different cluster IDs.

When a node announces its route, it attaches its cluster ID to the route. In this manner, nodes *outside* the area eventually see two different cluster IDs for the same effective destination area. We then need to modify HLR's route selection algorithm so that different partitions fall into different *subsets* (see subsection 3.2). Then, a node will pick one route for each partition correctly.

There are three other details to take care of. First, while nodes in an area converge on a cluster ID, the cluster ID visible externally might change, causing a fair bit of route churn. To reduce the churn, a node holds down a route that announces a change in cluster ID. Second, nodes within one partition of an area must be able to distinguish between routes to nodes within the same partition, and nodes from another partition of the same area. The latter routes might "enter" the partition from another area; HLR tags such external routes with the identifier of this external area in order to detect this. Finally, we must augment the route selection rules to prefer internal routes to external routes.

With these changes, HLR is able to route correctly without the assumption of internal connected-ness.

3.5 Discussion

In this section, we have described how HLIs can be leveraged to build scalable routing based on a variant of DSDV. Two questions arise when considering HLR in the context of sensor networks.

How does HLR interact with energy management schemes? In general, these schemes can be classified into two classes: topology control and coordinated sleep/wakeup. Topology control schemes try to maintain a connected network using a (continuously varying) fraction of the nodes. For such schemes, HLR should work without any change. For coordinated sleep-wakeup schemes, HLR will need to be slightly modified such that, if a node's next hop is currently asleep, it can buffer packets to that node until it awakes. With this modification, we believe that coordinated sleep/wakeup does not conceptually alter the correctness of HLR, nor does it impact its performance.

Many sensor network applications rely on nodes communicating with a base station: How does HLR fit in this scenario? It is conceptually possible to design a variant of HLR that supports this form of communication; we have left the design of this for future work.

4 Routing and Rendezvous Primitives

From a sensor network perspective, HLR enables a variety of routing and rendezvous primitives that can improve the scalability of systems like Directed Diffusion [7] or TinyDB [12], or enable data-centric storage systems like

GHT [21] even in the absence of location information. In this section, we show how HLR can be used to provide these primitives.

4.1 Unicast

HLR can provide "any-to-any" or unicast transmission primitives. More precisely, any node can send a message addressed to the HLI of any other node, and HLR attempts to deliver the message in a best-effort manner. Such a primitive can be useful in many contexts: monitoring the status of a node, or tasking a node to perform a specific action such as turning on a camera.

Achieving unicast functionality in HLR is rather straightforward. HLR forwards unicast packets based on the *longest prefix match* of HLI. However, HLR must allow a packet's address to match multiple routing table entries. This functionality enables correct packet delivery in the presence of network partitions (Section 3.4). As we have discussed earlier, when more than one entry matches, a separate copy of the packet is forwarded for each matching entry, *i.e.,* one copy of each packet is delivered to each partition of the destination area. To avoid multiple copies delivered to each partition, the destination area of every copy is associated with a partition cluster ID, *i.e.,* the destination of every copy of the packet is defined by the pair (destination area, cluster ID). Since the area HLI plus the cluster ID can uniquely identify a partition of the area, it is guaranteed that each partition will receive exactly *one* copy of the packet. All copies but one are dropped when they enter the lowest-level area; the partition that contains the destination node will correctly deliver the packet to the destination.

An alternative would have been to forward packets along one of the entries, and either back-track (which would involve maintaining state in the routing protocol) or have the node "tunnel" the packet to the partition containing the destination. Both these approaches are complex, and we chose to trade-off some additional overhead in packet duplication assuming partitions happen infrequently.

4.2 Area Broadcast and Area Anycast

HLR also provides two other powerful routing primitives: broadcasting to all nodes within an area, or anycasting to one node within the area. Thus, a broadcast packet addressed to an HLI prefix 1.2 would be delivered (best-effort, of course) to all nodes within that area. Similarly, an anycast packet (a bit in the packet header distinguishes between anycast and broadcast packets) addressed to a HLI prefix 1.2 would be delivered to *some* node within that area.

The implementation of these primitives falls out quite easily from HLR's basic design. An area anycast is forwarded similarly as a unicast packet until it reaches some node within the destination area. When an area is partitioned, it suffices to forward the area anycast towards one of the partitions. Finally, when a node receives an anycast packet whose HLI prefix is a prefix of its own HLI, it assumes that the packet is destined for itself.

An area broadcast is also forwarded much like a unicast packet until it reaches some node within the destination area. At that point, the packet is flooded throughout the area. Flooding within an area must be done with care. Consider a broadcast to area 1.2. If any node outside this area receives the packet from a node within the area, it drops the packet to prevent further propagation of the flooding. In the case of partitions, the broadcast packet must, of course, be delivered to all partitions.

We argue that these primitives will help scale data-centric routing protocols. In particular, because the areas are aligned along "application-specific" location boundaries (*e.g.,* in an in-building network, there might be areas corresponding to floors, and sub-areas corresponding to wings), we expect most location-based queries will also be well-aligned along area boundaries. Accordingly, we expect these primitives to be used fairly frequently in a sensor network deployment.

Finally, we believe it is also possible to implement source-specific multicast [3] using reverse-path forwarding on the routing table provided by HLR. We have left the design of this primitive to future work.

4.3 Rendezvous Based on Random Hashing

HLR also provides *rendezvous* primitives that can be used to implement data-centric storage schemes like distributed hash tables. For this, HLR basically provides a way to consistently and randomly hash an arbitrary key to a node in the network using a primitive called `hash-lookup(key)`. This primitive is similar in principle to the key lookup provided by distributed hash table (DHT) systems like CAN [20] or Chord [25], but its implementation is very different. Using this primitive, it is possible to implement the DHT primitives such as `put(key,packet)` and `get(key)`. Furthermore, using the lookup functions provided by `hash-lookup` it is also possible to implement other rendezvous mechanisms like the triggers proposed in [24]. We do not discuss the details of this implementation here, but note that such triggers can be very useful for actuation based on the occurrence of certain events within a sensor network.

Prior work [21] has proposed to implement these primitives using geographic routing. HLR can achieve similar functionality without using geographic routing. HLR provides this functionality by treating a hashed key as an

HLI, and routing the packet containing that key to the node whose HLI is closest to the key. Before we describe the details of the implementation, we must note that HLR's hashing does not necessarily maintain all the properties of DHTs. In a classical DHT, the key space is likely to be much larger (128 or 160 bits) than the HLI space. Furthermore, in a classical DHT, the nodes are arranged uniformly along the key space (enabling load balancing), while in HLR the node location in the key space is determined by the HLI assignment to nodes. To some extent, this can be rectified by carefully assigning HLIs since this assignment is under the control of the network administrator.

Function hash-lookup() sends a packet that has the key as the destination HLI. In addition, the packet has a bit indicating that it needs to be processed as hash lookup. Assume for a moment that the network has converged, the routing tables don't change, and the network is not partitioned. Then, every node in the system has one routing entry for each top-level area. The node that issues the `hash-lookup()` treats the key as an HLI and routes the packet to the top-level area whose area identifier is closest to but larger than the top-level area in the key (with wraparound). For example, assume that there are three top-level areas in the system: 1, 5 and 7. Then, the key 4.3.2 would first be routed towards area 5, by our rule, and a key 8.5.1 would be routed to area 1. When the packet reaches area 5, the same procedure is now followed, but at the second level of the area hierarchy, until a final node is reached.

In the presence of partitions, the cluster ID determines which partition is "closer" to the key.

This hashing algorithm has an interesting property: a node has enough local information to determine if it should be the target of a `hash-lookup()`. It can determine if its own top-level area is closest to the key, and so on recursively. This property is useful in maintaining the correctness of a `hash-lookup()`; if, because of routing transients a node receives a lookup not destined for itself, it can re-route the packet.

Implementing a distributed hash table using our primitive is simple. The `put()` and `get()` primitives can be implemented the same way as `hash-lookup()`. Local replication is then simply a matter of storing an additional copy at the node in the leaf area whose ID is the *second* closest to that of the corresponding area ID in the key. Triggers of the kind suggested by [24] can be similarly implemented.

4.4 Data-Locality Preserving Hashing

In the previous subsection, we have introduced a rendezvous primitive which is based on randomly hashing a specified key. A newly introduced data-centric storage scheme, DIM [11], uses a data-locality preserving hash. In this section, we show that HLR can be extended to support this kind of hashing as well. The basic idea is to map the multi-dimensional data space to HLIs so that each HLI is assigned a hyper-rectangle of the data space such that at any level, the hyper-rectangles assigned to all HLIs at that level disjointly cover the entire data space. Ultimately, every node is assigned a disjoint hyper-rectangle in the multi-dimensional data space, *i.e.,* the node *owns* the hyper-rectangle. In this section, we discuss how HLR can provide locality-preserving hashing, and how a simplified version of DIM can be built on top of it.

Concretely, we say that HLR provides a *data-space multicast* primitive `send-dsm(H,p)` which delivers packet `p` to all the nodes that own part of the hyper-rectangle `H`. (Of course, unicasting to a single point in the data-space is a degenerate case of this primitive, so we don't discuss it further. We have left an exploration of an analogous anycast primitive to future work.)

To understand how HLR implements the `send-dsm()` primitive, we need to describe how the hyper-rectangles in data-space are mapped to nodes. We use a mapping very similar to the one used in DIM [11], but instead of relying on geographic divisions, we divide the HLI space, as illustrated in Figure 3.

To show the basic idea, we take a 2-d space $[0,1) \times [0,1)$ as an example and map it to the network shown in Figure 2. Our description here can be easily generalized to multi-dimensional data spaces. At the top level, areas 1, 2, and 3 are divided into two sets which partition the data space aligned with the first dimension. The result is that area 1 is responsible for sub-space $[0,0.5) \times [0,1)$ and areas 2 and 3 together are responsible for sub-space $[0.5,1) \times [0,1)$. We repeat such divisions within each set of areas and alternatively aligned with each dimension of the data space until each set contains only one node. For example, areas 2 and 3 equally divide the sub-space $[0.5,1) \times [0,1)$ while the sub-space $[0,0.5) \times [0,1)$ is further divided among the sub-areas of area 1, and so on. Note that if the distribution of areas or the distribution of nodes within areas are not uniform, the division of the data space can be adjusted accordingly. For example, instead of choosing 0.5, we can use 0.1 or 0.9 for a dimensional division of the data space.

Given the mapping procedure above, it can be seen that the part of the data space assigned to each node is a hyper-rectangle in the data space and the hyper-rectangles of all nodes disjointly cover the entire data space. Furthermore, given the hierarchy in HLIs, an inherent property of our scheme is that the hyper-rectangles of HLIs which share the same prefix are also close in data space. Such a mapping enables the construction of a data-centric storage scheme that efficiently supports range queries.

Using this mapping between nodes and the data space,

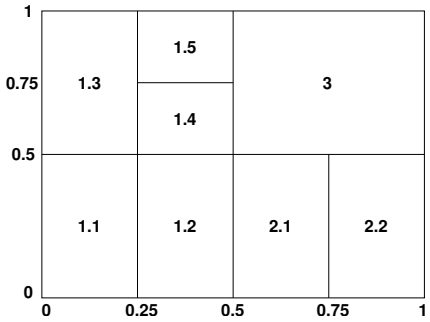

Figure 3: Example: mapping from 2-d data space to the network shown in Figure 2

how does HLR support the `send-dsm()` primitive? Given a hyper-rectangle `H`, each node can locally apply the above mapping procedure to determine which top-level areas might contain the nodes which would fall in `H`. Using this, the node at which `send-dsm()` is invoked will route the packets towards those top-level areas, creating copies of the packets if necessary (this is analogous to query splitting in DIM). This same procedure is applied recursively with each area until a copy of the packet reaches each node whose hyper-rectangle intersects `H`. At some point, when `H` entirely covers the hyper-rectangle associated with an area, HLR simply floods the packet within that area.

For example, assume a range query Q: $[0.6, 0.8) \times [0.3, 0.7)$ is issued at node 1.1.1. Node 1.1.1 looks it up in its routing table, and matches area 2 and 3 whose hyper-rectangle intersects Q. Therefore, node 1.1.1 will split Q into two sub-queries: Q_1: $[0.6, 0.8) \times [0.3, 0.5)$ and Q_2: $[0.6, 0.8) \times [0.5, 0.7)$, and send Q_1 to area 2 and Q_2 to area 3. When Q_1 reaches area 2, say node 2.1.1, it will be further split into two sub-queries: Q_{11}: $[0.6, 0.75) \times [0.3, 0.5)$ and Q_{12}: $[0.75, 0.8) \times [0.3, 0.5)$. The procedure goes on until the hyper-rectangle of the receiving node completely contains the sub-query. It is now easy to see how DIM can be built on top of `send-dsm(H,p)`. A DIM data insertion would specify a `H` which is merely a point (a degenerate case of a hyper-rectangle). For a DIM query, the `H` corresponds to the query rectangle itself. Query replies can simply be unicast to the HLI of the query issuer.

Our description of DIM on HLR has ignored dynamics such as node failure, node join, and link dynamics which may cause changes to HLR routing tables. When the routing table changes, the mapping between nodes and their hyper-rectangles might change. When this happens, a DIM built on HLR needs to check whether its hyper-rectangle has changed and whether the tuples it has stored need to migrate to some other nodes. As with random hashing, this check can be performed entirely locally. When a node decides that some of its data belongs to a hyper-rectangle that it no longer owns, it reinserts the data.[5] Finally, DIM's local replication can also be mimicked in HLR; recall that DIM stores an extra copy of the packet at a node that would have owned the hyper-rectangle if the current owner fails. Once area partition occurs, we need to *rebuild* DIM *locally* within the partitioned area by treating each partition a single sub-area. For example, when area 1.1 is partitioned due to some node failure, the hyper-rectangle mapped to area 1.1 is re-split among all partitions of area 1.1. In general, local rebuilding will cause data migration among sub-areas, but this overhead should be small assuming partitions happen relatively infrequently.

4.5 Summary

In this section, we have described how several routing primitives that are thought to be important for sensor networks can be supported using HLR. We have implemented *all* of these primitives (as well as simplified versions of distributed hash tables and DIM) in TinyOS. In the next section, we evaluate these primitives using simulation.

5 Performance Evaluation Through Simulations

In this section, we investigate HLR using simulations, comparing it to other methods of implementing the routing and rendezvous primitives (*e.g.*, using geographic routing). Now, it is easy to see that, asymptotically and with high enough density, *none* of HLR's primitives are likely to outperform a geographic routing based approach. In a dense network on a 2-dimensional surface, the asymptotic path lengths are $O(\sqrt{N})$, and geographic routing based approaches will approach this performance. Indeed, HLR will perform worse in general because route aggregation can increase path lengths. So, our real goal here is not to demonstrate that HLR is better than other alternatives, but that it is no worse than other alternatives. HLR's usefulness, then, is that it provides equivalent functionality while making fewer assumptions about available technology (*e.g.*, precise localization).

We perform four sets of experiments. First, we evaluate the performance of HLR unicast by comparing its average path length for all-pair communication with that of GPSR. Second, we compare the efficacy of area broadcast in HLR, by evaluating a workload of diffusion queries that are geographically scoped. We compare HLR against

[5] Note that this may happen often, as with a flapping route. A DIM built on HLR needs some hysteresis mechanisms built in that would prevent it from re-inserting data at every routing change. We have left the design of this for future work.

a version of Diffusion that uses a simple geocast mechanism [28]. Third, we implement a DHT and a DIM on HLR and compare them to GHT and DIM on top of GPSR for purpose of evaluating rendezvous primitives. Finally, we evaluate the performance of HLR under dynamics and measure the overhead induced network-wide by node failure.

5.1 Methodology and Metrics

We use *ns-2* for our simulations. We implemented HLR (including the functionality that detects and deals with partitions) in *ns-2*, and all of the routing and rendezvous-based primitives we described in Section 4. Using these primitives, we implemented a simplified version of one-phase pull Diffusion, GHT and DIM in *ns-2*. The total of HLR code for primitives and routing protocol is about 2800 lines.

An interesting methodological challenge we faced was to randomly generate connected hierarchical topologies for evaluating HLR. Our topology generator first computes random hierarchical areas where the depth of each area is random, and the size of the sub-areas is roughly same. Then it lays out this topology on a 2-dimensional surface.[6] To generate a random hierarchical topology, we first calculate the size of the network using the number of nodes in the network, radio range and density. (In our simulations, radios have a range of 30m. We also simulate for two different densities, 10 neighbors per node, and 20). Then we split the network into grids such that the number of grids is the smallest number greater than the total number of nodes. Now starting from the top-level areas, we randomly allocate contiguous free grids to this area, such that the number of grids equals the total number of nodes within the area. Then in a breadth first way, each sub area is allocated contiguous grids from its grids allocated to its parent area. This breadth-first approach can lead to unsatisfiable states, at which point our generator back-tracks and repeats the re-allocation procedure. Finally, we randomly pick a point within the grid as the coordinate of the node. We generate topologies whose size ranges from 25 to 200 nodes with step size of 25 nodes.

Unless otherwise specified, our metrics are: a) the *messaging cost* of implementing a particular primitive (for unicast, this can be equivalently expressed as the average path length), and b) the *control overhead* of HLR routing.

For most of our experiments, we compute the scaling behavior of the metric discussed above. We computed our metrics for several topologies ranging from 25 to 200 nodes. For each topology size, the reported number is an average of 5 randomly chosen topologies.[7]

[6]The reason we only use 2-D topology is simply because GPSR currently only works on 2-D. HLR doesn't rely on this assumption.

[7]Resource constraints prevented us from averaging over more topologies. Recall that our topology generation algorithm employs a back-tracking procedure to assign areas to node locations. Using our implementation, it sometimes took more than a day to generate an instance of the topology.

5.2 Results

Unicast Routing Performance Our first experiment simply measures the cost of unicast communication in HLR, and compares it with the cost of unicast using GPSR. For both these schemes, we conducted a simulation where each node sends a message to all of the other nodes. We then calculated the average path length incurred using either scheme. Figure 4 plots the average path length for HLR and GPSR. This figure shows that the average path length in HLR is often three hops longer than that in GPSR. These results are for a network with a density of 20, so GPSR, in most cases, does not incur perimeter mode routing. In a network with a density of 10 (figure not shown), the gap between GPSR and HLR is decreased to about one hop.

While this might seem somewhat pessimal, our current understanding of sensor networks suggests that they are not likely to be used for arbitrary point-to-point routing. Rather, we expect other primitives like rendezvous and area broadcast will be more likely used, since they more naturally support querying and triggering. Thus, we now discuss the performance of data-centric routing and data-centric storage systems implemented on HLI.

Diffusion We implemented a simplified version of one-phase pull Diffusion[8] in *ns-2*. This version uses two underlying routing layers, HLR and GPSR. We augmented GPSR to support *geocast* (broadcasting to all nodes within a rectangle). In our implementation, the packet is unicast using GPSR until it reaches a node within the specified region, and then flooded within the region.

Our goal in this experiment was to try to understand the expected performance of Diffusion on these two routing layers. Lacking traces of actual workloads, we generated a synthetic query workload for Diffusion. We generated interest messages of varying geographical scopes, assuming that the scopes were all aligned with the HLR areas. This assumption is not particularly disadvantageous for GPSR, but is also the most likely kind of query in an HLR based system (queries with un-aligned scopes can be implemented as multiple area broadcasts). We assume that the size of the geographic scope is distributed exponentially: most queries are to small areas.

Again, this seems like a plausible assumption for sensor network query workloads.

Figure 5 plots the comparison of average query delivery cost between Diffusion using GPSR and Diffusion over HLR. In this case, we assume that the query asks for the

[8]Equivalently, we can be said to have implemented the tree-building procedure that TinyDB [13] uses.

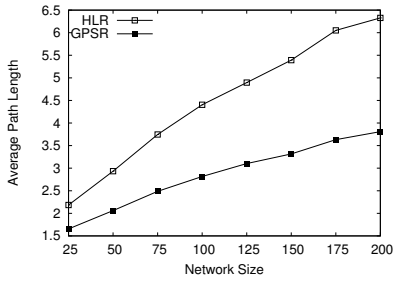

Figure 4: Comparison of average path length in GPSR and HLR on networks with density 20

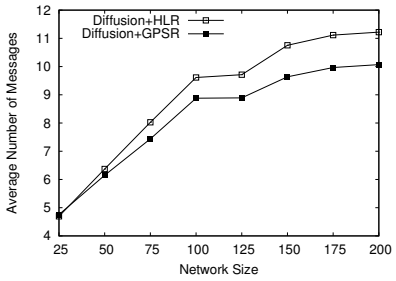

Figure 5: Comparison of average query cost in Diffusion on networks with density 20

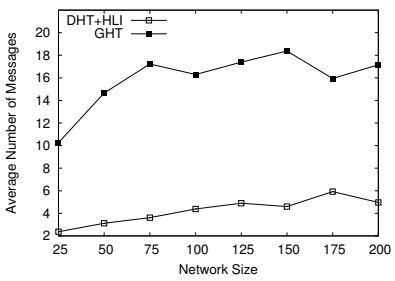

Figure 6: Comparison of average query cost in DHT over HLR and GHT on networks with density 20

`min` over a set of sensor readings at each node within the target areas. The results are aggregated along the return path.[9] Notice, in this case, that the performance of HLR is much closer to that of GPSR than was the case for all-pairs unicast. Clearly, in this case, the longer path lengths resulting from aggregation matter less, and the flooding costs dominate. This conclusion is true even at a lower node density (10), the results of which we omit for brevity.

GHT How well does a DHT implemented on HLR perform compared to a GHT? To test this, we performed several random `hash-lookup()`s on the DHT over HLR, and performed the equivalent `put()` operations in a GHT. Figure 6 compares the messaging cost of these two schemes.

In this particular case, we find that performance of DHT over HLR is *much better* than that of a GHT, quite unexpectedly given our results from Figure 4. The reason is simply because nearly every `put()` operation in a GHT incurs perimeter traversal, which is pretty expensive compared to greedy mode delivery. Further, some operations incur a traversal of the outer perimeter, which skews the average. In HLR, however, the average cost of a `hash-lookup()` is the same as the average unicast cost. For a lower density of 10 neighbors per node, the plots look almost identical (omitted for brevity). At these densities, DHT over HLR encounters longer paths, but GHT encounters longer perimeters as well.

DIM To evaluate the efficiency of the `send-dsm()` primitive, we implemented DIM on top of HLR and compared it to DIM op top of GPSR. For this comparison, we used sensor data collected from a deployed in-building testbed; each sensor periodically collects light, temperature and humidity readings. In the dataset, there were 509765 readings. From these readings, we generate a balanced insertion workload (10 insertion per node) for every node in the network from the data set. And we inserted the selected data subset into the DIM. For our query workload, we generated a set of 3-D range queries where the query box size is exponentially distributed and its location is uniformly placed.

Figure 7 plots the comparison of data insertion cost between two versions of DIM on networks with density 20. In most cases, DIM on HLR has smaller insertion cost than DIM on GPSR. In the latter, the existence of empty zones [11] forces DIM to rely on GPSR's perimeter mode to find the owner, resulting in a longer delivery path and a higher cost. At a lower density (10 neighbors per node), this performance advantage decreases. In HLR, paths become longer. However, DIM relies less on perimeter mode than GHT (see above), hence DIM is less affected by a decrease in density.

Finally, Figure 9 compares the query delivery cost between two versions of DIM on networks with density 20. Here again, we see that DIM on HLR outperforms DIM over GPSR. There are two contributors to this. One is, as before, that DIM on GPSR encounters many more perimeter traversals in discovering empty zones. The other is a more subtle point that has to do with the way the data-locality preserving hashes for the two schemes work. In DIM over GPSR, with 3 or higher dimensional data, a query hyper-rectangle may actually be split across two nodes that are far apart physically. However, in DIM over HLR, the query hyper-rectangle owned by an area is always enclosed within the hyper-rectangle belonging to the parent area. Thus, DIM over HLR preserves data-locality more than DIM over GPSR, explaining the performance improvement. At a lower density as shown in Figure 8, the performance difference is a little more, since some of the performance advantages come from the data-locality properties of HLR.

Dynamics Finally, we address the important question of HLR performance under network dynamics. Specifically, we are interested in HLR overhead caused by the

[9]Therefore, for each query in Diffusion, query delivery cost equals reply delivery cost.

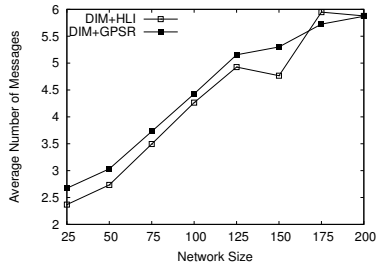

Figure 7: Comparison of average insertion cost in DIM on networks with density 20

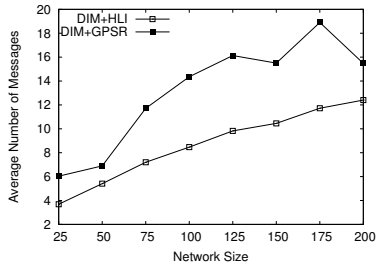

Figure 8: Comparison of average query cost in DIM on networks with density 10

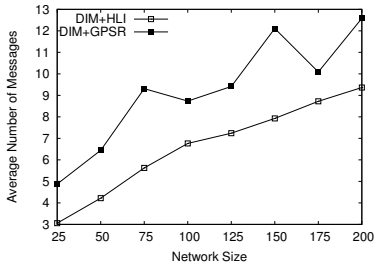

Figure 9: Comparison of average query cost in DIM on networks with density 20

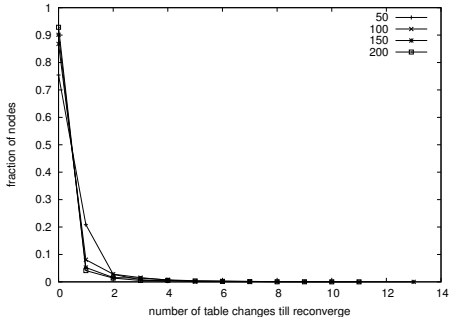

Figure 10: Average number of routing table changes under single node failure on networks with size 50, 100, 150, 200 and density 20.

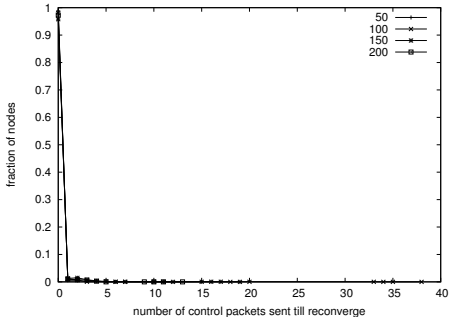

Figure 11: Average number of control packets to re-converge under single node failure on networks with size 50, 100, 150, 200 and density 20.

failure of a single node. In our experiment, we sequentially fail and recover each node in the network, waiting long enough for the network to re-converge between node failures. Our two metrics for HLR performance are: a) the average number of routing table changes caused by a single node failure, and b) the number of routing messages sent until the network converges after a single failure. For each network size, we computed these metrics over five instances of topology sizes.

Figure 10 plots the distribution of routing table changes. This figure shows how well HLR localizes the effect of failure, an important consideration for wireless sensor networks. On average, more than 90% nodes are unaffected by a node failure! Only when nodes at the boundary of top-level areas fail do we see that some nodes change their routing tables several time before convergence. Even then, the magnitude of these changes is relatively small even for networks of 200 nodes.

Figure 11 plots the distribution of extra overhead caused by a single node failure. Again, this value is noticeably small. In most cases, the vast majority of the nodes are completely unaffected by a single failure and see no routing traffic at all for a failure. In the most egregious cases, some nodes see about 20 messages while the routing protocol converges.

From our perspective, this is highly encouraging; the impact of dynamics is very local and is one of the bigger selling points of HLR.

6 Implementation

We have implemented the HLR routing protocols and most of the routing and rendezvous primitives in on Berkeley motes. Figure 12 shows the software architecture of our implementation. The NeighborList module of TinyDiffusion exports a filtered send and filtered receive interface which filters out bad quality links including asymmetric links and fragile links. On top of this, as discussed in Section 3, currently we just use a simple hop-count as our path metric.

The HLR core module implements the core algorithm of HLR which constructs and maintains the routing table. The routing table management module helps organize the routing table by effective destination area in order to enable efficient route processing. The routing primitive module implements unicast, area multicast and area anycast, while the rendezvous primitives module implements the hashing lookup function and data-preserving hashing function. As we have described in Section 4, the hashing lookup function and data-preserving hashing make use of the routing primitives for data delivery.

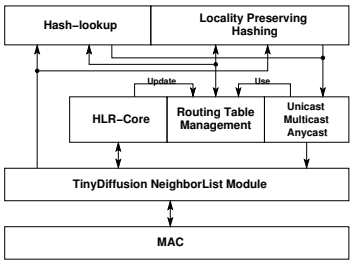

Figure 12: HLR Software Architecture

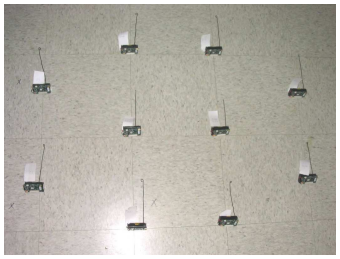

Figure 13: HLR Experiment Topology

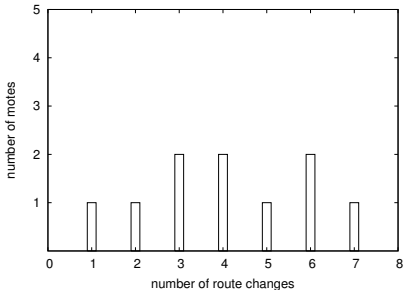

Figure 14: HLR Experiment Result with the topology shown in Figure 13

As a proof-of-concept, we ran HLR on a network of 10 Mica-2 motes. The topology for our experiment is shown in Figure 13 where the mote transmission power has been reduced in order to create a multi-hop network. We let HLR run for four hours. Figure 14 gives the number of routing table changes during that four hour interval. As we can see, over four hours in the worst case, a mote saw *seven* routing table changes indicating that the routing tables might be expected to be quite stable in realistic deployments. However, many aspects of HLR need to be verified in the real world: partition recovery, better path metrics, and dynamics in larger deployments,

7 Conclusion

In this paper, we have described a pragmatic routing layer for sensor networks. This layer is built upon the observation that many sensor network nodes will be assigned hierarchical location identifiers. We described the design of HLR, a routing protocol that constructs scalable routing tables. Using HLR, it is possible to implement several routing primitives for data-centric routing and storage. Our results indicate that HLR performs well and contains dynamics. We intend to experimentally validate these aspects of HLR using our mote implementation.

References

[1] N. Bulusu, D. Estrin, L. Girod, and J. Heidemann. Scalable Coordination for Wireless Sensor Networks: Self-configuring Localization Systems. In *Proceedings of the Sixth International Symposium on Communication Theory and Applications (ISCTA '01)*, Ambleside, Lake District, UK, July 2001.

[2] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proceedings of the 9th Annual International on Mobile Computing and Networking*, San Deigo, CA, September 2003.

[3] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The pim architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, 1996.

[4] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks . Technical report, Internet draft, July 2002.

[5] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, San Diego, CA, September 2003.

[6] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00)*, January 2000.

[7] 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, August 2000.

[8] 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, August 2000.

[9] L. Kleinrock and F. Kamoun. Hierarchical Routing for Large Networks: Performance Evaluation and Optimization. *Computer Networks*, 1:155–174, 1977.

[10] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Computer Networks: The International Journal of Computer and Telecommunications Networking, Special issue: Wireless sensor networks*, pages 499–518, November 2003.

[11] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multidimensional Range Queries in Sensor Networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, November 2003.

[12] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor for Sensor Networks. In *Proceedings of ACM SIGCMOD*, San Diego, CA, June 2003.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGregation Service for Ad-Hoc Sensor Networks. In *Proceedings of 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[14] J. Moy. *RFC 2328: OSPF Version 2*, April 1998.

[15] J. Newsome and D. Song. GEM: Graph Embedding for Routing and Data-Centric Storage in Sensor Networks without Geographic Information. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, November 2003.

[16] G. Pei and M. Gerla. Mobility management for hierarchical wireless networks. *Mobile Networks and Applications*, 6(4):331–337, August 2001.

[17] C. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the ACM SIGCOMM*, London, UK, August 1994.

[18] R. Ramanathan and Martha Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications*, 3(1):101–119, June 1998.

[19] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing wihtout Location Information. In *Proceedings of the 9th Annual International on Mobile Computing and Networking*, San Deigo, CA, September 2003.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM*, San Diego, CA, August 2001.

[21] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash Table for Data-Centric Storage. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[22] Y. Rekhter and T. Li. *RFC 1771: A border gateway protocol 4 (BGP-4)*, March 1995.

[23] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.

[24] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of the ACM SIGCOMM*, Pittsburgh, PA, August 2002.

[25] 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*, San Diego, CA, August 2001.

[26] P. F. Tsuchiya. The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks. In *Proceedings of the ACM SIGCOMM*, Stanford, CA, August 1988.

[27] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, November 2003.

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