GEM: Graph EMbedding for Routing and Data-Centric Storage in Sensor Networks Without Geographic Information

James Newsome Dawn Song

March 2003
CMU-CS-03-112

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper we introduce GEM (Graph EMbedding for sensor networks), an infrastructure for node-to-node routing and data-centric storage and information processing in sensor networks. In GEM, we construct a labeled graph that can be embedded in the original network topology in an efficient and distributed fashion. In that graph, each node is given a label that encodes its position in the original network topology. This allows messages to be efficiently routed through the network, while each node only needs to know the labels of its neighbors.
To demonstrate how GEM can be applied, we have developed a concrete graph embedding method, VPCS (Virtual Polar Coordinate Space), which embeds a ringed tree into the network topology. We have also developed VPCR, an efficient routing algorithm that uses VPCS. VPCR is the first algorithm for node-to-node routing that guarantees reachability, requires each node to keep state only about its immediate neighbors, and requires no geographic information. Our simulation results show that VPCR is robust on dynamic networks, works well in the face of voids and obstacles, and scales well with network size and density.

This research was supported in part by the Center for Computer and Communications Security at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, by NSF Aladdin Center grants CCR-0122581 and CCR-0058982, and by gifts from Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, ARO, Bosch, Carnegie Mellon University, or the U.S. Government or any of its agencies.

Keywords: Sensor networks, routing, data-centric storage

1 Introduction

The widespread deployment of sensor networks is on the horizon. Networks of thousands of miniature sensors may present an economical solution to some of our challenging problems: real-time traffic monitoring, safety monitoring (structural, fire, and physical security monitoring), military sensing and tracking, measurement of seismic activity, real-time pollution monitoring, wildlife monitoring, wildfire tracking, etc. However, these sensor networks introduce new research challenges. On one hand, large sensor deployments could provide an unprecedented volume of measurements across wide areas. On the other hand, because such deployments are likely to consist of devices with limited computation, memory, storage, communication range, and most importantly, battery power, many traditional tools and techniques do not apply to sensor networks. The lack of tools and techniques to enable communication among sensor nodes and retrieval of useful information from sensor networks limits the use of sensor networks and impedes large sensor network deployments.

One crucial problem in sensor networks is how to retrieve sensed data. Previous solutions can be classified into three categories [31]: local storage, external storage, and the data-centric approach. In local storage, each node keeps the data it senses locally. To retrieve data in local storage, a query must be flooded through the network, causing nodes with data relevant to the query to send data back to the base station. In external storage, data is sent to the base station without waiting for a user to send a query. While external storage saves having to flood the network with a query, it may waste energy when data that the user is not interested in is sent to the base station.

In the third approach, data-centric storage[31, 34], events are named, and sensors cooperate locally to detect named events, such as elephant sightings. When a node detects a named event, it determines what node is responsible for that name, and then stores the data at that node. Which node is responsible for storing a type of data is typically determined by taking a hash of the name, and mapping that hash onto a node in the network. When a user wishes to query the network, he can send the query only to the node or nodes responsible for the data relevant to the query. Note that in this approach, queries do not need to be flooded through the network, nor does data that the user does not ask about get sent to the base station. Additionally, the query may be partially processed at the nodes storing the data, allowing a small message consisting of aggregated data to be sent to the base station instead of all individual records relevant to the query.

Data-centric storage was first explored in distributed hash tables designed for internet use, such as CAN[31], Chord[35], Pastry[33], and Tapestry[38]. These systems are not suitable for sensor networks, because routing is done through an *overlay* network. Each hop in the overlay network can be several hops in the actual network. As a result, it is necessary to have an underlying routing mechanism in addition to the overlay routing. More importantly, packets may cross the physical network multiple times, wasting energy and bandwidth[1]. The only previous system which provides data-centric information processing and aggregation in sensor networks is GHT [31]. Instead of using an overlay network, GHT uses the geographic coordinates of the sensor motes to map them onto a 2-dimensional distributed hash table. It uses a geographic routing algorithm, GPSR [18], to route from one node to another. Geographic routing has several desirable properties: nodes can route to each-other knowing only the coordinates of their immediate neighbors, it is possible to perform broadcasts constrained to a geographic region, and dealing with node failures is largely trivial. Unfortunately, such an approach may not apply to many sensor networks. First, this

[1]There has been work in constructing overlay networks close to the network topology, such as [30], which could alleviate this effect.

approach requires each sensor node to know its *exact* geographic location. Current methods of determining geographic location[3, 7, 10, 12, 28] consume much energy and may not be possible in many sensor network scenarios. Second, GPSR works best when geographic locality accurately represents network topology. For many sensor networks, geographic locality may differ significantly from network topology. For example, physical obstacles can easily prevent two geographically close nodes from communicating directly, causing them to be far apart in the network topology. Third, GPSR's perimeter-mode forwarding does not work in three-dimensional topologies, such as may be found in an office building.

In this paper we propose Graph EMbedding for sensor networks, or *GEM*. The GEM framework provides an efficient infrastructure for data-centric information processing and storage by jointly addressing the issues of routing and mapping from data names to nodes. In GEM, we label the sensor nodes in a distributed and efficient manner such that (1) Nodes can efficiently route messages addressed to labels, knowing only the labels of their neighbors, and (2) we can efficiently map data names to labels, in order to do data-centric storage. By using graph embedding and a node labeling method, GEM enables efficient node-to-node communication and data-centric aggregation and storage.

GEM is a new systematic approach to provide infrastructure support for node-to-node communication and data-centric applications in sensor networks. Our approach does not rely on geographic information. When geographic information is imprecise, or when geographic locality differs from the network topology, our approach has significant advantages over previous work such as GPSR [18] and GHT [31].

To demonstrate how GEM can be applied, we have developed *VPCS* (Virtual Polar Coordinate Space), a graph embedding technique that embeds a virtual polar coordinate space onto the network topology. The VPCS works correctly when constructed without any regard to physical layout, though we have developed two techniques to improve performance by aligning the virtual space with the network topology. The first scheme requires nodes to find the distances between themselves and their neighbors, but the second scheme (which we show to be more effective) does not require this capability. We have also developed *VPCR* (Virtual Polar Coordinate Routing), an algorithm for routing within the virtual polar coordinate space. We could build data-centric applications such as data-centric routing, data-centric storage, and data-centric aggregation using VPCS, and enable general node-to-node communication using VPCR. VPCS and VPCR are extremely light weight. Our experiments and evaluations show that VPCR is extremely efficient in terms of both total energy usage and hotspot energy usage.

Our contributions are as follows:

- We introduce the GEM framework, in which a graph is labeled in a manner to provide efficient routing and data-centric storage, and is then embedded in actual sensor network topologies.

- We have developed VPCS, an instance of GEM that embeds a ringed tree into the network topology, using a virtual polar coordinate system. VPCS is efficient to construct and is robust to network dynamics such as node failures and additions.

- We have developed VPCR, a routing algorithm that makes use of VPCS. VPCR is the first node-to-node routing algorithm that requires each node to keep state only about its immediate neighbors and requires no geographic information. VPCR routes packets along nearly the shortest path, adapts to network dynamics well, and scales to large networks.

We begin by further examining graph embedding in Section 2. Next we describe an instance of GEM, VPCS, in Section 3. In Section 4 we describe how VPCS can be used to enable VPCR, an efficient node-to-node routing algorithm. In Section 5 we show how VPCS heals itself after node failures and accommodates node additions. In Section 6 we evaluate VPCS and the routing performance of VPCR, and show that they scale well to increasing network sizes and densities, and continue to work in the face of network dynamics and irregular topologies. Next we cover a few remaining issues and potential areas of future work in Section 7. In Section 8 we discuss related work. Finally, we conclude in Section 9. Acknowledgments to those who have helped make this work possible are in Section 10.

2 The GEM Framework

Graph embedding is a technique in graph theory in which a *guest* graph $\mathcal{G}$, is mapped into a *host* graph $\mathcal{H}$. It has been applied to many different problems[32]. In particular, it has been used to map one interconnection network on another[4, 13, 19, 20], e.g. to simulate one topology using another. In GEM, we use graph embedding to take a network topology that is convenient to work with, and map that onto real network topologies.

Graph embedding is defined in [32] as follows. An *embedding* of the graph $\mathcal{G}$ (the *guest* graph), consists of two mappings: (1) The *node-assignment function* α maps the set of nodes in $\mathcal{G}$ one-to-one into the set of nodes in $\mathcal{H}$. (2) The *edge-routing function* ρ assigns to each edge $\{u, v\} \in \mathbf{E}(\mathcal{G})$ a path in $\mathcal{H}$ that connects nodes $\alpha(u)$ and $\alpha(v)$.

In GEM, we apply the idea of graph embedding to sensor networks in two steps. First, we must choose a labeled guest graph $\mathcal{G}$ that can be used for efficient routing and data-centric storage. The second step is to embed the guest graph $\mathcal{G}$ on the actual sensor network topology, $\mathcal{H}$.

Choosing a Guest Graph The first step is to choose a guest graph that we can use for efficient routing and data-centric storage. The graph should have the following properties:

- Routing: The labeled graph $\mathcal{G}$ should have the property that it enables nodes to route messages efficiently from one label to another label when each node only knows the labels assigned to its neighbors. Thus, the labels of nodes implicitly contain information about the network topology.

- Mapping for distributed hash table: We can develop a function f that efficiently maps a key k to a label in the label space L.

- Low embedding overhead: We must be able to embed the graph $\mathcal{G}$ into the topology $\mathcal{H}$ in a distributed manner, and without incurring unnecessary overhead. For details, see the embedding step, below.

- Fault tolerance: Nodes in the sensor network will fail- either the guest graph must be able to route around failed nodes, or it must be feasible to dynamically reconstruct the graph embedding to *repair* the guest graph.

The second step is to develop an embedding to efficiently simulate the guest graph $\mathcal{G}$ using the actual network topology, $\mathcal{H}$. There are several metrics for graph embeddings[32], but the one we will focus on optimizing is *dilation.* To put it simply, we wish for the edge-routing function ρ to map the edges in $\mathcal{G}$ to paths that are as short as possible in $\mathcal{H}$. Ideally, each edge in $\mathcal{G}$ would simply be mapped to a corresponding edge in $\mathcal{H}$. However, when the appropriate connection does not exist in $\mathcal{H}$, the edge must be simulated by a path.

We show that using graph embedding in this manner could serve as an underlying infrastructure and enable many applications efficiently:

Enabling Data-Centric Storage To support data-centric storage, a data item of a certain name k, can be mapped to a label $f(k)$ and then routed to and stored at the node with that label. A querier for data of that name can likewise compute $f(k)$ and send a query to the same node. Because the labeled graph $\mathcal{G}$ is embedded in the connectivity graph $\mathcal{H}$, routing in the label space can be almost as efficient as shortest-path routing in the original network topology graph. This is significantly different from the distributed hash table work for distributed systems over the Internet, as we mentioned in the introduction.

Enabling Node-to-Node Routing To enable general node-to-node routing, we can implement a lookup mechanism to find a node's current label using data-centric storage. Assume that a node has a permanent identifier n, and its label is $\mathcal{L}(n)$. Node n can store its label in the distributed hash table, at the node corresponding to $f(n)$ in the same manner that it would store a piece of data. When another node wishes to communicate with n, it can send a lookup request to the node with label $f(n)$, retrieving n's label $\mathcal{L}(n)$. Once the node has n's label $\mathcal{L}(n)$, it can address messages to $\mathcal{L}(n)$ and they will be routed through the embedded graph to n.

3 VPCS (Virtual Polar Coordinate Space)

In order to demonstrate how graph embedding can be applied in sensor networks, we have designed and developed Virtual Polar Coordinate Space, or VPCS. In VPCS, we embed a *ringed tree*[2] graph into the network topology. We do this by assigning each node a level, which is the number of hops to the root of the tree, and a virtual angle range, which uniquely identifies a node within a level. We also develop efficient mechanisms which enable us to assign the virtual angles to be consistent with the network topology. As a result, the labels can be thought of as defining a virtual polar coordinate space. In Section 4 we show that this property enables us to use greedy forwarding to achieve efficient routing.

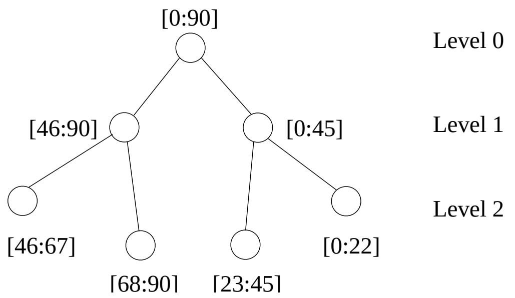
3.1 Building the Virtual Polar Coordinate Space: The Basic Scheme

The first step to build the virtual polar coordinate space, or VPCS, is to embed a ringed tree. Essentially, we can follow the well known algorithm to build a spanning tree. The cross-links to make the tree a *ringed* tree largely take care of themselves[3].

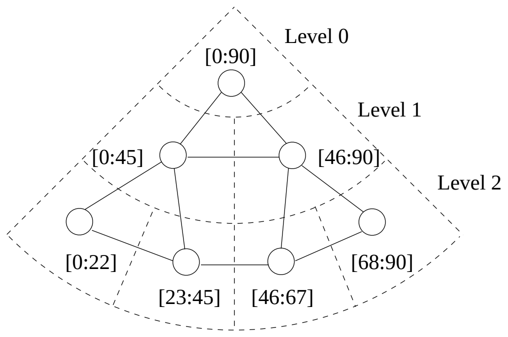
To build the spanning tree, we first choose a root node. Any node may perform this role, though a base station would be a logical choice if there is one present. The root broadcasts a message stating that it is at level 0 of the tree. Any nodes that are within radio range of the root become children of the root. These nodes then each broadcast a message advertising that they are at level 1 of the tree. The root node hears these messages and marks the senders as its children. Nodes that haven't joined the tree yet that hear these messages make the sender their parent. If a

[2]The ringed tree is a type of *x-tree*. X-trees and ringed trees were first proposed in [9]. An x-tree is a tree with extra connections added for more efficient and load-balanced routing. In the ringed tree, cross-edges are added to connect "adjacent" nodes of the same level: siblings and "cousins".

[3]Nearby nodes of the same level can form cross-links, though some nodes that would have cross-links in a proper ringed tree will not be within radio range of each other.



(a) Simple angle range assignment



(b) Ordered angle range assignment, with cross-links

Figure 1: Sample angle range assignments.

node hears more than one such message, it may pick any of the senders as its parent. One logical choice in this case is to choose the parent whose message was received with the greatest signal strength. This process continues recursively until all nodes reachable from the base station have joined the tree.

After the tree is built, information about the size of each subtree is propagated back upward towards the root. Leaf nodes initiate this by reporting to their parent a subtree size of one. When a node has received a message from each of its children reporting its subtree size, it adds one for itself and reports its subtree size to its parent.

Once the root of the tree has received the subtree size of each of its children, it has enough information to begin assigning virtual angles. To begin, the root is assigned the entire angle range being used. In a geometric polar coordinate system this would be 0 to 2π. However, we wish to be able to subdivide the range many times without dealing with fractions, so a range such as 0 to $2^{16} - 1$ or 0 to $2^{32} - 1$ is more appropriate.

The root then assigns each child a subset of its range. The size of the subset assigned to each child is proportional to the size of that child's subtree. For example, if the children have subtrees of sizes 10, 20, and 15, they should be assigned $\frac{10}{45}$, $\frac{20}{45}$, and $\frac{15}{45}$ of the angle range respectively. This balancing gives larger (wider) subtrees a wider angle range to match. It also helps to ensure that the angle space is not exhausted before reaching all the leaves of the tree. Each child then assigns each of its children a subset of its angle range, continuing recursively until every node is assigned an angle range. For an example of this numbering, see Figure 1(a).

The energy cost for this construction is fairly low. All together, each node must send three messages during the construction of the virtual space. Each node sends a message when building the tree, another when propagating the size of the subtrees, and a third when assigning virtual angle ranges.

3.2 Consistency Requirements

In order for VPCS to function correctly, the following conditions must hold:

1. Each node must have a parent (except the root node).

2. Each node must be assigned a level equal to its parent's level plus one.
3. Each node must be assigned a virtual angle range that is a subset of its parent's virtual angle range.
4. No node may have two children with overlapping angle ranges.

It is clear that the algorithm in Section 3.1 satisfies the above consistency requirements. As we will later show, these conditions enable effective routing and data-centric storage.

3.3 Aligning VPCS with Network Topology Without Geographic Information

The method described so far for assigning the virtual space is sufficient to provide reliable node to node routing, as described in Section 4.1 and Section 4.2. However, in order to take advantage of the extra *cross*-links in the ringed-tree structure, we need to label the nodes in such a way that a node can know which cross-link it should use, if any. To accomplish this, we attempt to align the virtual coordinate space more closely with the network topology. Specifically, if one were to follow a "ring" along a level of the tree, the angles should be strictly increasing or decreasing (until they wrap around). For an illustration of this, see Figure 1(b). We show in Section 4.3 how this allows us to use the cross links to perform more efficient routing.

To perform this alignment in a distributed fashion, each parent must determine the order of its children in the ring. There are potentially many ways to do this; we describe two ways below. The naive scheme requires nodes to be able to find the distances between themselves and their neighbors, and use that information to build a local coordinate system. The improved scheme does not rely upon any localization assumptions, and achieves better results.

Naive Scheme: Using distance information First we explore the case where nodes can estimate the distance between themselves and their neighbors. Some previously proposed range finding techniques are by observing radio signal attenuation[3], or by measuring the arrival time difference between radio and ultrasonic pulses[28]. Each node can then use this information to determine the coordinates of each of its neighbors in a local coordinate system, as described in [8]. It can then put its children in order by geometric angle.

Unfortunately, as we show in the evaluation, even small errors in the distance estimates prevent the children from being ordered correctly. Additionally, subtrees may occasionally cross, in which case it would make more sense to consider the angle to the child's entire subtree, rather than just the angle to the child itself.

Improved Scheme: Using a global coordinate system We now describe how to use a *global* coordinate system to solve both these problems, and how one can be built without localization data.

We build a global coordinate system by using triangulation from the root and two other reference nodes. The root node uses some simple heuristics to select the other reference nodes such that the three nodes are not colinear, and are not too close together. In order for each node to calculate its position, it needs to know its distance to each reference node, and the distances between the reference nodes. Instead of measuring distances directly as in the local coordinate system method, we measure distances as the number of network hops on the shortest path between nodes. We first build a temporary spanning tree from each of the two reference nodes. Every node then knows its distance in network hops from each reference node: it is equal to the node's level in the reference node's spanning tree. One of the reference nodes can then send the distance between itself and the other reference node to the root node. The root node then floods the network with the distances in network hops between itself and the two other reference nodes. Using this information, each node can calculate its position in the network.

Note that because the distances are measured in network hops, the resulting position estimates are not highly accurate, and have a coarse granularity. This makes it impractical for nodes to order their children by the angles to children themselves. In fact, because of the low granularity of this method, some siblings are likely to have exactly the same coordinates. We solve this problem by sorting the children by the angles to their *centers of mass*. The center of mass can be thought of as the point in the "center" of a node's subtree. We define a node's center of mass as the average of the coordinates in that node's subtree, including its own. A leaf node's center of mass is simply its own coordinates. A non-leaf node's center of mass is the average x and y coordinates of the centers of mass of each of its children, weighted by the sizes of their subtrees. Sorting children by the angles to their centers of mass helps solve the problem of inaccuracy, because the errors in each node's individual coordinates will tend to average out in the center of mass. It also helps solve the problem of low granularity, because siblings' centers of mass will be further apart in the coordinate system than the nodes themselves.

In Section 6, we show that this scheme, which *uses no geographic information* performs better than the naive scheme, which assumed that nodes could find the distances to their neighbors.

4 VPCR: A Routing Algorithm for VPCS

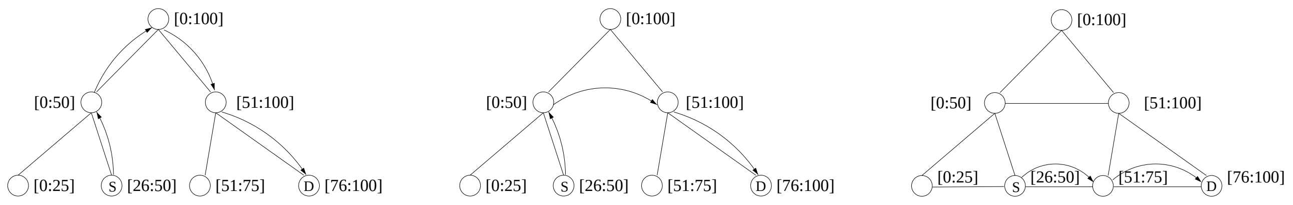
In this section we describe VPCR, a routing algorithm built on VPCS. VPCR routes from any node to any point in the VPCS, where a point is defined by a level and angle. In order to simplify the description of VPCR, we first describe the *naive-tree* routing algorithm, which does not use the cross-links of the ringed tree structure. We next discuss *smart-tree* routing, which is obtained by some simple optimizations to the naive-tree algorithm. Finally, we describe *Virtual Polar Coordinate Routing* algorithm (VPCR), which uses greedy forwarding to take advantage of the ringed tree's cross-links.

All three algorithms are guaranteed to be correct so long as the underlying VPCS meets the consistency requirements listed in Section 3.2. Each successive algorithm introduces performance optimizations over the previous one.

4.1 Naive Tree Routing

In previous work, spanning trees have been used to perform node to base station routing[21]. In these algorithms, a spanning tree is built with the base station at the root. For a message to be routed to the base station, each node passes the message to its parent until it reaches the base station. In this manner, a message can be sent to the base station from any node in the network, using the minimum number of hops.

Using VPCS, this algorithm can easily be extended to accomplish node to node routing. Each message is first routed up the tree until it reaches an ancestor of the destination. It is then routed down the tree to the destination. At each hop, the current node can determine whether the destination is in its subtree by checking whether the destination virtual angle is within its angle range, and if the destination level is greater than the current level. If not, then the destination is not in the node's subtree, so the packet is forwarded to the node's parent. If the destination is in the node's subtree, it uses the same method to determine which child's subtree the destination is



(a) Naive Tree, 4 hops (b) Smart Tree, 3 hops (c) VPCR, 2 hops

Figure 2: A packet is routed from S to D using the three routing algorithms. Smart Tree and VPCR use a 1-hop neighborhood.

in, and forwards the message to that child. This procedure is illustrated in Figure 2(a).

Note that in this algorithm, once a packet reaches an ancestor of the destination, the path from that ancestor to the destination is the shortest path possible. The inefficient part of the algorithm is in having to route up the tree to reach an ancestor of the destination. Hence, the paths can be made shorter if we can avoid routing up the tree as much as possible, and instead route laterally through the tree to get to an ancestor of the destination. Not only will this make the paths shorter, it will also prevent nodes high in the tree from becoming traffic hot spots.

4.2 Smart Tree Routing

The first optimization we describe is the smart tree algorithm. In cases where the naive tree algorithm says to route up the tree, the smart tree algorithm first checks nearby nodes to see if any of them are an ancestor of the destination, or the destination itself. If so, it forwards to that node. In Figure 2(b), the smart tree algorithm saves a hop by doing this.

This method can be further improved if each node keeps state about its neighbors' neighbors, or 2-hop neighbors. In this case, instead of only checking its set of neighbors for an ancestor of the destination, a node can check its set of 2-hop neighbors. If a suitable destination is found that is 2 hops away, the packet can be forwarded to one of the neighbors that can reach the 2-hop neighbor.

In the more general case, each node can maintain state about all nodes up to n-hops away. We will refer to this set of nodes as a node's *neighborhood*. Routing performance can be expected to improve substantially with larger neighborhoods. However, the amount of state that must be stored and maintained at each node grows exponentially with n. In the extreme case that the entire network is in each node's neighborhood, a packet will always be routed along the shortest path, but every node is storing the topology of the entire network.

4.3 Virtual Polar Coordinate Routing

The smart tree optimization only provides shortcuts once the packet reaches a node that is near an ancestor of the destination. For nodes that are far away, it is still likely that a packet will need to be routed several hops up the tree before it reaches a node that can take advantage of the smart tree optimization. VPCR attempts to use the cross-links in the ringed tree to route laterally through the tree, even when the current node doesn't explicitly know about an ancestor of the destination. It does this by assuming that the virtual angles are assigned to be strictly increasing or decreasing along the ringed tree's rings. This ordering can be accomplished as described in Section 3.3. When this is the case, each node's label can be thought of as defining a space in a virtual polar coordinate system, as illustrated in Figure 1(b).

The Virtual Polar Coordinate Routing algorithm, or VPCR, uses this polar coordinate space for more efficient routing. When the smart tree algorithm would be forced to route up the tree, VPCR checks to see if any nodes in its *neighborhood* (see Section 4.2) have an angle range that is closer[4] to the destination angle than the current node's angle range. If so, the packet is greedily forwarded closer to the destination angle range. Figure 2(c) shows how this can achieve significantly shorter path lengths than the naive tree or smart tree algorithms.

Sometimes the packet will reach a local minimum, where no nodes in the current neighborhood have a closer angle range to the destination than the current node. This happens when there is a cross link missing in the embedded ringed tree[5]. When this happens, we can use the tree structure to find a *path* to use in place of the missing link. In practice, we accomplish this by simply routing to the node's parent when the packet is at a local minimum. Eventually the packet will reach a node where the cross-link is not broken, and greedy forwarding can continue, or it will reach an ancestor of the destination, in which case it can simply be routed back down the tree to the destination.

If the VPCS is well aligned with the network topology, the result will be that packets will follow a curve from the source to the destination. For short angular distances, such a curve is not much longer than a straight line. However, for angles larger than about 115°, the distance to travel to the origin and then to the destination (i.e. up the tree to the root, and then back down), is shorter than such a curve. Therefore in our current implementation we forward up the tree in such cases rather than trying to follow the curve.

Correctness An important property of this algorithm is that as long as the VPCS is consistent (as defined in Section 3.2), packets will always reach their destinations. When a node using VPCR cannot find a nearby node whose angles are closer to the destination angle, it routes to its parent. This will always cause it to eventually reach an ancestor of the destination, at which point it can follow the tree down to the destination.

Loop-free Another important property of VPCR is that routing loops are impossible. At any particular routing step, the packet is *never* routed to a node whose angle range is further from the destination angle than the current range. It will only greedily forward to a node whose angles are *closer* to the destination angle than the current node's angles. When it fails to find such a node, it routes to its parent. Because the parent's angle range is a superset of the current node's angles, it is no further from the destination angle than the current node. Note that the parent cannot forward the packet back to the child, because the child's angles can be no closer to the destination than the parent's.

5 Dynamic Issues in VPCS

The topology of a sensor network may change over time. In particular, nodes may fail, and new nodes may be added. When either of these things happen, the VPCS must be modified to remain consistent.

[4]We define the distance between an angle range R and an angle A to be the distance between A and the angle in R that is closest to it. If A is contained in R, then the distance is zero.

[5]The link may be missing either because there is physically a break in the ring- two consecutive nodes in the ring aren't close enough to have a link, or because nodes were assigned angles in the incorrect order (see Section 3.3)

When making changes to the VPCS, the consistency requirements in Section 3.2 must be enforced. In addition to these requirements, there are two goals to keep in mind when modifying the VPCS: stability, and alignment with the network topology.

Stability Depending on how node to node communication is used, virtual coordinate changes can be expensive. In the case of data-centric storage, virtual coordinate changes can lead to having to move stored data from one node to another. Additionally, the coordinate changes themselves require communication with every node whose coordinates must change, which uses energy. Therefore, it is desirable to make as few changes as possible to the VPCS while keeping it consistent.

Alignment with Network Topology When routing using VPCR, the more closely the VPCS reflects the network topology, the more effective its greedy forwarding will be. If changes to the VPCS damage this relationship, VPCR routing performance will suffer to some degree. Hence, it is also desirable to preserve any correlation between the VPCS and the network topology.

Unfortunately, it is difficult to achieve both of these goals. One way to preserve the alignment with the network topology would be to 'reboot' some or all of the network. That is, starting from some point sufficiently high in the tree, erase all parent child relationships and addresses assigned. Then, begin the algorithm described in Section 3.1 to rebuild that subtree. This is likely the best way to keep the VPCS coordinates correlated with physical space, but it requires many nodes to be assigned new virtual coordinates. As mentioned above, this is expensive in itself, and can also be expensive for applications running on top of VPCS. Because of this expense, this method is unlikely to be acceptable except in cases where the topology changes very infrequently. In that case, improved routing performance may offset the cost of having to rebuild part of the VPCS when the topology does change.

The algorithms we have implemented put more emphasis on minimizing the number of nodes that must be assigned new addresses than on preserving the correlation between VPCS and physical space. As a result, relatively few nodes should have to change their addresses after the failure or addition of a node, but the correlation between VPCS and physical space may degrade after repeated changes.

5.1 Dealing with Node Failures

Individual sensor nodes tend to be unreliable. The most common reason for failure is likely to be energy depletion, though some may fall prey to a hostile environment. In any case, for VPCS to be a practical solution, it must be able to deal with failures with minimum overhead.

When a node P fails, its children are left without a parent, violating consistency requirement 1 (See Section 3.2). In the simplest case, each orphaned child C can find another node P' that is connected to the tree to use as a parent. When C makes P' its parent, consistency requirement 3 will be violated, because P''s angle range will not contain C's angle range. To fix this, P' must add C's angle range to its own. The parent of P' must do the same. This continues up the tree up to the lowest common ancestor between the failed node P and the new parent P'.

At this point, the lowest common ancestor between P and P' has two children with overlapping angle ranges (an ancestor of P, and an ancestor of P'), violating consistency requirement 4. To fix this, it takes C's angle range away from the child that was an ancestor of the failed node P. That node must do the same, onward back down the tree until the parent of P removes the angle range from its own[6].

[6]In some cases, this will result in a node having no angle range left. When this happens, the parent of the node with no angle range must take away some of the angle range from one of its other children's subtrees and give it to

Last, in order to satisfy consistency requirement 2, the (previously) orphaned child C must set its level to the level of its new parent P', plus one. Its children must do the same, recursing the change through its subtree.

Sometimes an orphaned child will not be able to reach a connected node to make its new parent, but one of its descendants will be able to. In this case, that descendant must reverse the portion of the tree between itself and the orphaned child. That is, its parent must become its child, and that reversal must continue until it reaches the orphaned child. While changing the parent-child relationships, the angle ranges must be kept consistent. To do this, the connected descendant (now the root of the disconnected subtree) takes the angle range of the orphaned child. It then assigns its (previous) parent the new angle range, minus the angle ranges of its other children. The (previous) parent follows the same procedure. Again, this process continues until it reaches the orphaned child[7].

Once the tree reversal has been performed, the new root of the disconnected subtree makes the connected node its parent, and performs the rest of the reconnection algorithm as already described.

The last case possible is that no nodes in the disconnected subtree can reach a connected node. In this case the subtree is partitioned from the network, and there is physically no way to rejoin it.

Issues There are a couple potential problems with the algorithm as described so far. The first problem is that some nodes may end up with discontinuous angle ranges. For example, if a node has the range 50-100, and the range 60-70 is taken away, it is left with 50-59 and 71-100. While this is acceptable for VPCS and VPCR, it makes the storage of neighbor angle ranges more complex. The angle range for any one node could be made up of any number of discontinuous sets.

The other problem is that during the tree repair algorithm, another node involved in the algorithm may fail. Unless care is taken in the implementation of the repair algorithm, this could cause the VPCS to become inconsistent. It may be possible to use something akin to a two phase locking algorithm, though the details of this are left as future work.

If the angle ranges become heavily fragmented, or if the VPCS becomes inconsistent, the affected subtree can simply reboot itself. That is, the parent-child relationships can be deleted, and the angles and levels assigned to each node forgotten. The root of the rebooted subtree can then simply rerun the initial setup algorithm to put that part of the tree back into a fresh state.

5.2 Adding Nodes

For many applications, it is desirable to be able to add new nodes into the network. Some reasons for adding new nodes are to replace failed nodes, to improve sensor coverage area, or to incorporate new types of sensors.

In order for a new node to become part of the VPCS, it must join the tree and be assigned a level and angle range. As with failures, we have chosen to make the process of adding new nodes inexpensive rather than emphasizing keeping the virtual coordinate space consistent with physical coordinates.

When new nodes are added to a VPCS network, they must join the routing tree and be assigned a level and angle range. In order to do this, the new node first chooses a parent from its set of neighbors. The parent assigns the new node a level equal to its own plus one. The parent then assigns the new node an angle range by first taking away part of the angle range from one of its

the node.

[7]This step can also leave a node with no angle range. When this happens, some angle range must be taken away from a sibling's subtree.

other children. That child must take away that angle range from its child that it is assigned to as well. Thus, this change recurses down the tree to a leaf node[8].

If the new node is able to find a connected node to use as a parent, and can also reach a node that was previously partitioned from the network, the previously disconnected node may connect via the new node. In this case, the new node becomes its parent. The new node assigns angle ranges to any children gained in this manner in the same way as when the tree is first built: it attempts to put the children in order by angle, and then assigns each a subset of its angles, in sizes proportional to each child's subtree.

6 Evaluation of VPCS and VPCR

In this section we will evaluate the performance of the routing algorithms that we have presented.

Simulation Parameters The parameters used for our experiments are listed in the following table:

Range	40 m
Network Size	400 m x 400 m
Density	Avg Degree 15
Nodes	509
Neighborhood	2 hops

In order to show that VPCR is effective for routing in large networks, we chose a network size relatively large in relation to the nodes' radio range. A packet sent along one side of the network would necessarily have to travel at least 10 hops. A packet sent from one corner of the network to the other would have to travel at least 15 hops. We chose a network density such that each node has an average of 15 neighbors. This ensures that randomly generated topologies tend to be well connected.

We set the neighborhood size to be 2 hops for the smart tree and VPCR algorithms. Recall from Section 4.2 that the neighborhood is the set of nearby nodes that each node keeps routing information about. Larger neighborhoods improve routing performance at the cost of additional state. We show in Section 6.9 that a 2 hop neighborhood provides a good balance between routing performance and the amount of state that must be maintained at each node.

Simulation Procedure We obtained these results using a custom simulator developed by the authors. The simulator assumes an ideal radio model and an ideal MAC layer; if two nodes are within radio range, they can communicate without packet loss. All results were obtained by running the test on 5 randomly generated topologies and averaging the results. In each topology, each node is placed randomly except the root of the tree, which is placed at the center of the network[9]. When a randomly generated topology had fewer than 95% of the nodes reachable from the root, it was thrown away and replaced by another.

In our VPCR simulations we used both the local coordinate system and global coordinate system

[8]Another possibility is for each node to *reserve* some angle space during the initial setup, so that if it gains a new child it doesn't have to take any space away from another child. However, we did not examine this approach in detail because the reserved space will cause a data-centric storage system to be biased towards low-level nodes in the tree.

[9]We have tested the routing algorithms that we have described to verify that they function correctly regardless of where the root of the tree is placed. However, placing the root at the center of the network minimizes the depth of the ringed tree that is built, which improves the performance of tree-based routing. Because VPCR uses the ringed tree's cross-links, this choice has a smaller impact on its performance than on the performance of the Naive and Smart Tree algorithms.

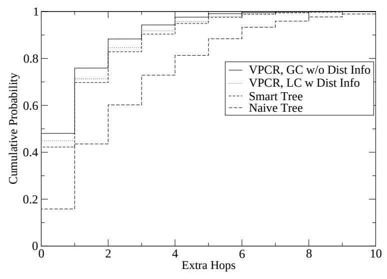


Figure 3: CDF of extra hops over the shortest path.

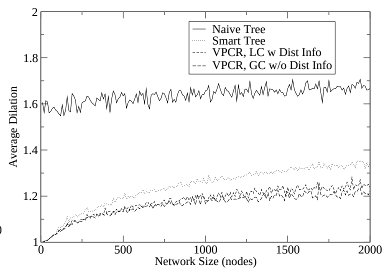


Figure 4: Dilation as network size is increased.

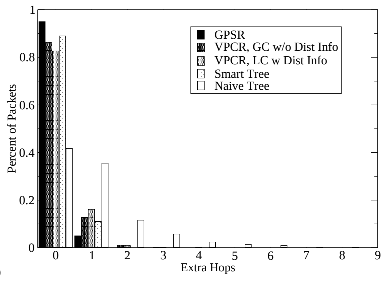


Figure 5: Histogram of extra hops taken over shortest path.

methods, described in Section 3.3. In the results labeled "**VPCR, LC w Dist Info**" we use the naive scheme of Section 3.3, which assumes the nodes can find the distance between neighboring nodes, and uses that information to build a local coordinate system at each node. In the results labeled "**VPCR, GC w/o Dist Info**" we use the improved scheme, which does *not* assume that nodes can find the distance between neighboring nodes. Instead it uses selected reference nodes to build a coarse grained global coordinate system. For these experiments we place two reference nodes at the edge of the network, 90 degrees apart.

6.1 Near Shortest Path Performance

Figure 3 shows the cumulative distribution function for how many extra hops each packet takes over the shortest possible path. Note that using the global coordinate method of ordering, VPCR routes 75% of its packets with 0 or 1 extra hops. Even without using the ringed tree's cross-links, the smart tree algorithm routes 70% of its packets with 0 or 1 extra hops.

6.2 Scalability to Large Networks

Next we examine how well VPCR scales with the size of the network. For this test, we increased the number of nodes while holding the density constant. Because the average path length grows as the network gets bigger, we measure the average *dilation* rather than the number of hops over the shortest path. The dilation of a packet is defined as the number of hops actually taken divided by the number of hops on the shortest path. Figure 4 shows the average dilation in networks from 10 to 2000 nodes. Because all the variations of VPCR are tree-based, the overhead grows logarithmically with the size of the network.

6.3 Comparison with GPSR

Figure 5 compares the routing algorithms we have developed with GPSR. This figure shows what fraction of packets travel how many additional hops over the shortest path. The test parameters and GPSR results were taken from [18]. The network is 2250 m by 450 m, and contains 112 nodes having radio ranges of 250 m. It is important to note that in the GPSR simulation, nodes were moving using the random waypoint model, while in our simulations nodes remained stationary. We used the results with the least mobility for this comparison, in which the pause time was 120 s.

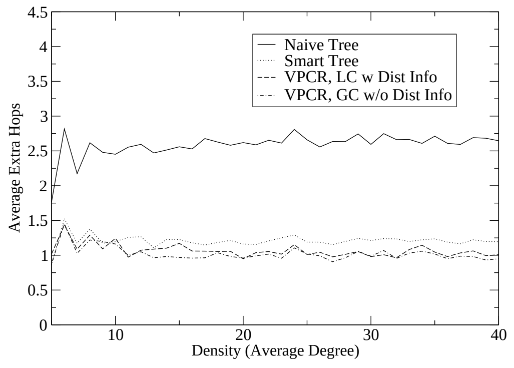


Figure 6: Average extra hops for varying network density.

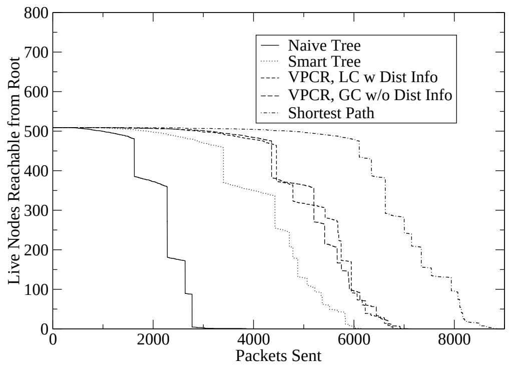


Figure 7: Network Lifetime

All but the naive tree algorithm do quite well here. One interesting thing to note is that the smart tree algorithm does slightly better than VPCR for these network parameters. This is because the network is not much wider than the node radio range. Thus, a message being forwarded up the tree is likely to reach a node in the neighborhood of the destination before having to go up to an ancestor of the destination. It is difficult for VPCR's greedy forwarding to do much better than this, and occasionally it will make a mistake. Even so, very few packets travel more than one extra hop over the shortest path for either algorithm.

6.4 Scalability to Network Density

Figure 6 shows how the different algorithms perform at different network densities. Here the network density is measured in average degree, which is the average number of neighbors each node has. For this experiment the size of the network was kept constant at 400 x 400 meters, and the number of nodes was varied to alter the density. This experiment shows that network density has little effect on the performance of VPCR or its variations.

6.5 Load Balancing

One problem with the naive tree algorithm is that it tends to route a lot of packets through nodes high in the tree. Given that each node has a finite amount of energy, this is likely to cause those nodes to become exhausted fairly quickly, resulting in the root of the tree becoming cut off from the rest of the network. The smart tree and VPCR algorithms attempt to refrain from routing up the tree as much as the naive tree algorithm, so those algorithms should not suffer from this problem quite as much.

In order to evaluate VPCR's load balancing, we routed packets to and from random nodes as in earlier tests, but allowed each particular node to forward only 200 packets before failing. Figure 7 shows the number of live nodes reachable from the root of the tree as packets are sent through the network and nodes fail. As expected, the nodes reachable in the naive tree network drops off quite rapidly as the nodes high in the tree fail. The smart tree and VPCR networks last considerably longer, though neither lasts as long as a network where all packets are routed along the shortest path. In future work, VPCR could be modified to incorporate energy awareness, which could provide better load balancing than shortest path routing.

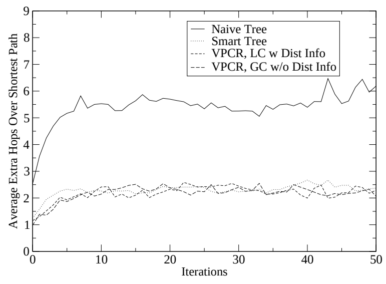


Figure 8: Average extra hops as nodes fail and are replaced.

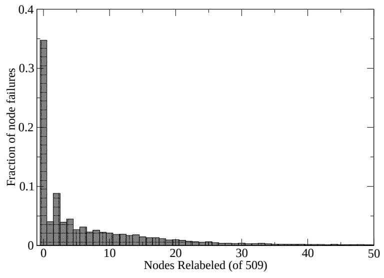


Figure 9: Histogram of number of nodes relabeled after a node fails.

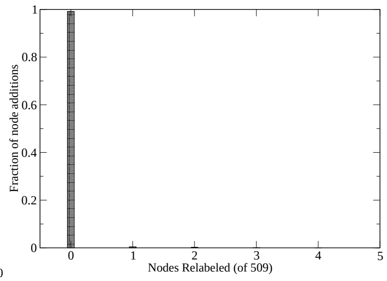


Figure 10: Histogram of number of nodes relabeled after a node is added.

6.6 Adaptability to Dynamic Networks

Adding and removing nodes tends to adversely affect the tree in several ways. As the angle space is rearranged to keep the tree consistent, the virtual polar coordinate space may become less consistent with the network topology. Additionally, when a node chooses a new parent, it tries to minimize the number of nodes that have to be reassigned virtual polar coordinates, which may result in branches of the tree going in odd directions and overlapping one another.

In order to determine the effects of repeatedly adding and removing nodes from the tree, we performed the following experiment. Starting with a freshly built tree, we first randomly choose 20% of the nodes to fail. After performing the tree repair algorithm, we measure the average number of extra hops of packets routed in that network. We then randomly place the same number of new nodes in the network and have them join the tree. The average number of extra hops is measured again at this point. This process was repeated for 50 iterations. The results are shown in Figure 8.

Routing performance degrades for the first 10 iterations, at which point it levels off. This is because our current repair and addition algorithms put more emphasis on making repairs cheap than on keeping the VPCS aligned with the network topology. This means that the correlation between physical coordinates and the virtual polar coordinates breaks down after repeated repairs. Additionally, many failures at once can result in voids in the network. This causes the tree to have to 'wrap around' these voids. Even when new nodes are added to fill these voids, these abnormalities in the tree remain.

In order to verify that the repair and addition algorithms are light-weight, we also measured how many nodes had to be relabeled during this experiment. Figure 9 shows a histogram of the number of nodes that needed to be relabeled after each failure. 35% of the node failures didn't require any nodes to be relabeled to repair the tree. The median number of nodes that needed to be relabeled was 3, though there were outliers in the data where up to the entire network needed to be relabeled. Figure 10 shows a histogram of the number of nodes that needed to be relabeled to accommodate each new node added to the tree. 99% of the node additions didn't require any nodes to be relabeled. The most nodes that had to be relabeled for any addition was 13.

For applications where there are many packets being routed, and nodes fail relatively infrequently, it may be desirable to use a more heavy-weight repair algorithm to ensure that routing stays as efficient as possible.

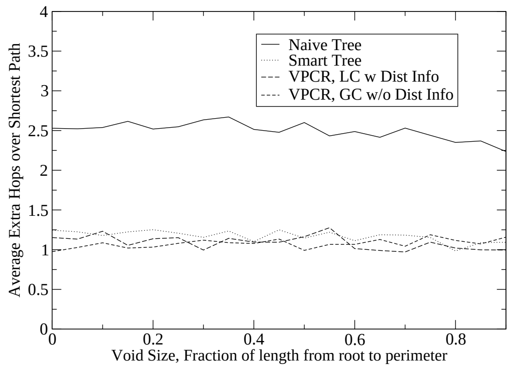


Figure 11: Performance in the presence of a void that extends radially through the network.

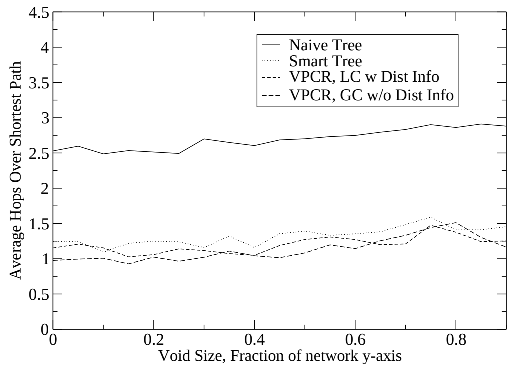


Figure 12: Performance in the presence of a void that extends vertically from one side of the network to the other.

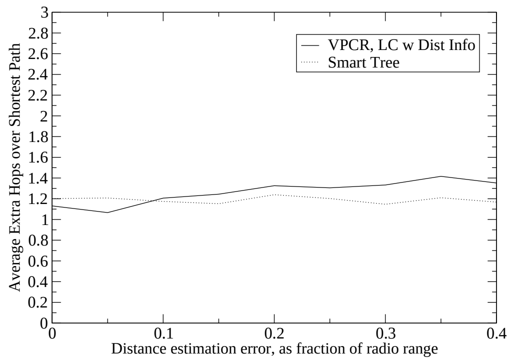


Figure 13: VPCR performance using local distance information vs the accuracy of that information.

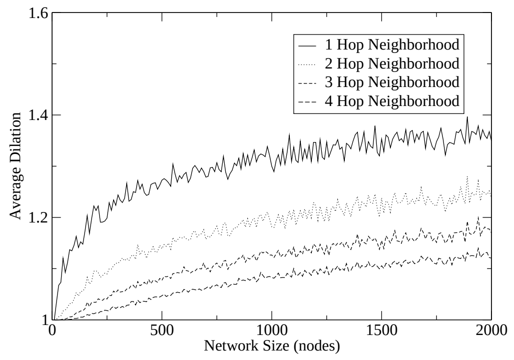


Figure 14: Dilation when different size neighborhoods are used.

6.7 Irregular Network Layouts

In many real applications, sensor nodes will not be deployed evenly and in an open area. There can be barriers blocking radio transmission, or equivalently, large voids. In order to test VPCR's performance in the presence of these irregularities, we simulated two types of voids. The first type of void goes in a line from the center of the network (the root), outward towards the perimeter. Figure 11 shows VPCR's performance as this void grows until it reaches the perimeter of the network. This type of void has very little effect on the performance of VPCR.

Next we simulated a void that is parallel to the y axis of the network. Figure 12 shows VPCR's performance as this void grows until it reaches both edges of the network. Performance degrades slightly here, because this type of void adversely affects how the spanning tree is built.

6.8 Effect of Inaccurate Distance Information on Routing in Scheme I

When using distance information to build local coordinate systems (scheme I in Section 3.3), inaccurate distance information leads to inaccurate coordinate systems, which can lead to the VPCS not being well aligned with the network topology. Figure 13 shows how well VPCR performs when the local coordinate system method is being used, for varying accuracies of distance estimates. The fraction of error here is the fraction of the radio range. Since we are using a radio range of 40m, an error of 10% means that estimated distances are randomly off by up to 4 m. The results of this simulation show that VPCR's greedy forwarding is beneficial until the error gets worse than 10%. At that point, greedy forwarding makes enough mistakes such that the smart tree algorithm tends to yield shorter paths.

Note that this issue is irrelevant to scheme II, which does not assume that nodes can find the distances between each other.

6.9 Choosing the Neighborhood Size

Figure 14 shows the effect of different size neighborhoods on VPCR. Each additional hop added to the network size appears to give diminishing returns. While the 2-hop neighborhood performs much better than a 1-hop neighborhood, 3 and 4 hop neighborhoods provide a relatively small improvement. This suggests that a 2-hop wide neighborhood may be optimal, since it provides the greatest performance improvement for the least additional state at each node.

7 Discussion and Future Work

7.1 Load Balancing for Data-centric Storage

When using VPCS for data-centric storage, it is important to balance where data is stored among the nodes. The simplest goal is to design the mapping function from data name to node such that one node is as likely as any other to be assigned. In VPCS, we map a name to a node by hashing the name, and mapping that hash to a level and angle. The data is then assigned to the node that covers that point in the virtual space. If there is no node at that level and angle, then the highest level node that covers that angle is responsible for the data. For proper load balancing to occur, each node should have a similarly sized piece of the virtual space.

We achieve this goal in two ways. First, there are fewer nodes at low levels of the tree than high levels of the tree. We offset this by biasing the hash function such that the probability of a level being chosen is proportional to the number of nodes at that level. The bias can be determined by actually counting the number of nodes at each level of the tree, though an estimate will often be good enough. Second, when assigning angles, nodes with larger subtrees get a proportionally larger piece of the angle space. This causes a problem because nodes with large subtrees get a larger piece of the virtual space, and are therefore more likely to be assigned data to store. However, this problem is somewhat offset because a node with no children takes responsibility for any data that maps to its angle range and level greater than or equal to its own.

7.2 Node Mobility

In some sensor networks, sensors may move over time. The simplest way to handle this is to use the current VPCS healing mechanisms. When a node loses contact with its parent, the child can treat the parent as having failed, and use the algorithm in Section 5.1 to recover. This method should be sufficient if nodes move slowly or infrequently enough that neighbor relationships don't change often. It is also possible that, in some scenarios, most nodes will remain stationary and only a few nodes are mobile. When this is the case, the mobile nodes can refrain from taking on children. This makes the repair mechanism very cheap when a mobile node moves; the parent of the mobile node simply deletes that node as a child, and the mobile node can find a new parent at its new location.

7.3 Improved Routing in the Ringed Tree

The ringed tree graph type has been studied in graph theory and in system architecture. It is meant to be used for load balanced, failure tolerant, and efficient routing. We are currently researching the ringed tree itself more closely in hopes of finding a routing algorithm that better achieves these goals. In particular, if the routing algorithm itself were tolerant to node failures, we may not need to perform the tree repair algorithm described in Section 5.1. If such an algorithm is found, we can use graph embedding to more closely model the ringed tree structure in VPCS, and thus gain these benefits.

7.4 Multiple Trees

Because VPCR is tree-based, nodes near the root of the tree tend to have to route more traffic than other nodes. This imbalance can be alleviated by using several smaller spanning trees rather than one large one. In this case, each node's label would consist of a tree identifier in addition to a level and virtual angle range. Within each tree, each node would have to know which trees border its own. In order to determine how to route a message to a neighbor tree, we can use data-centric storage within the tree to map tree id's to the labels of nodes that border those trees. In order to route to a node several trees away, it may be feasible for every node to know the higher level topology and use it to determine the shortest path. For example, suppose there is a 5000 node network consisting of 10 trees of 500 nodes each. It is not unreasonable for each node to store the neighbor relationships for 10 trees, though it would obviously be infeasible to store the neighbor relationships for 5000 individual nodes.

Another possibility is to have n trees, each with different roots, that each span the entire network. That is, each node would belong to all n trees. When a node wishes to send a message to another node, it could pick any of these trees to route the message through. This would help alleviate the hot spot at the roots of the trees. However, it still needs to be worked out how this method would interact with data-centric storage. One possibility would be for the mapping function f to specify a tree in addition to a label (see Section 2). However, this would not utilize the full benefit of multiple trees, since data and queries of a particular name would always be routed through the same tree.

7.5 Other Graph Types

It may be beneficial to investigate other graph types. A tree is useful because its hierarchical structure makes it easy to design a light weight routing algorithm that is guaranteed to be correct. Unfortunately, the hierarchical nature tends to overload nodes that are at low levels of the tree. Another possibility may be to use a grid. A grid would do away with the hierarchical nature, which could mean better load balancing. However, a grid would be more difficult to construct without geographic information, and guaranteed routing correctness would be more difficult, if not impossible.

8 Related Work

Routing Algorithms Rao et al. independently proposed another algorithm for performing node to node routing with only neighbor information, without geographic location information[29]. This paper describes an algorithm in which a virtual coordinate system is built by having nodes on the perimeter of the network determine their positions relative to each other. They then use an iterative relaxation algorithm for other nodes to determine their coordinates. Once the coordinate system is built, they use greedy forwarding to perform all the routing. This approach has the advantage that node failures and node mobility are more easily dealt with. It also has demonstrated routing performance on par with geographic routing where the physical coordinates are known. However, there is a relatively large set-up overhead for the perimeter nodes to find their relative positions, and to perform several iterations of the relaxation algorithm.

There has also been work on informative labeling schemes. The idea of labeling each node such that the distance between two nodes could be determined using only their labels was proposed in [24]. In particular, a distance labeling scheme for n-vertex trees was proposed, using $O(\log^3 n)$ bit labels. Compared to our labeling mechanism, the distance information given by this method is more precise, at the cost of labels whose length grows with the size of the network. The idea of labeling schemes that carry various types of information was further examined in [25]. Several papers have explored tradeoffs in routing algorithms between label size, memory requirements at each node, and routing performance[1, 2, 11, 23, 36].

Several papers have proposed other solutions to routing in wireless sensor networks. Many of these do not address node to node routing. Instead they assume that the only communication necessary will be nodes sending results to the base station, and the base station flooding the network with queries. Examples of this type of algorithm are Minimum Cost Forwarding[37], and LEACH[14]. Other routing algorithms have been proposed for sensor networks which do support node to node routing, but assume that sensors can find their precise locations. The first such algorithms proposed are GPSR[18] and GEDIR+FACE2[5]. More general ad-hoc routing protocols could also be used in sensor networks, but they have a cost associated with establishing and maintaining routes. The costs in terms of state kept at each node and routing update messages are too high to scale well in a large sensor network. Broch et. al. provide a survey of several ad hoc routing protocols[6], including DSDV[26], TORA[22], DSR[16][17], and AODV[27].

Data Dissemination Several data dissemination protocols have also been examined. [34] introduces the concept of data-centric storage, as well as describing the simpler concepts of local and external storage. This work is continued in [31], which describes GHT, a data-centric storage system for sensor networks built on top of GPSR[18]. Directed diffusion[15] and TAG[21] are more advanced forms of external storage which incorporate some in-network aggregation of data.

9 Conclusion

In this paper we proposed Graph Embedding for sensor networks. GEM can be used to solve a number of problems in sensor networks, including data-centric storage and general node to node routing. We have demonstrated how the concept of graph embedding can be used to build a Virtual Polar Coordinate System in a sensor network, and how Virtual Polar Coordinate Routing can be used to efficiently route within a VPCS. VPCR is the first solution to node to node routing and data-centric storage that requires each node to keep state only about its neighbors, and does not require geographic information. It is robust to dynamic networks, works well in the face of voids and obstacles, and scales well with network size and density. We hope that our graph embedding approach provides a new perspective for building sensor network applications.

10 Acknowledgments

This research was supported in part by the Center for Computer and Communications Security at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office. The authors would like to thank Brad Karp, Adrian Perrig, and Srini Seshan for their discussion and feedback. The authors would also like to thank Bruce Maggs for his valuable help and input on graph embedding. Finally, the authors would like to thank the shepherd Ramesh Govindan, and the anonymous reviewers, whose feedback has helped us to improve this paper.

References

[1] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, **11**(3):307-341, 1990.

[2] B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Mathematics*, **5**(2):151-162, 1992.

[3] P. Bahl and V.N. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. *IEEE Infocom*, 2000.

[4] S.N. Bhatt, F.R.K. Chung, J.W. Hong, F.T. Leighton, B. Obrenic, A.L. Rosenberg, and E.J. Schwabe. Optimal emulations by butterfly-like networks. *Journal of the ACM*, **43**:293-330, 1996.

[5] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999.

[6] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *International Conference on Mobile Computing and Networking*, 1998.

[7] Nirupama Bulusu, John Heidemann, and Deborah Estrin. *GPS-less low cost outdoor localization for very small devices.* 00–729. University of Southern California, April 2000.

[8] Srdan Capkun, Maher Hamdi, and Jean-Pierre Hubaux. GPS-free positioning in mobile ad-hoc networks. *Hawaii International Conference on Systems Sciences*, 2001.

[9] Alvin M. Despain and David A. Patterson. X-tree: a tree structured multi-processor architecture. *5th International Symposium on Computer Architecture*, pages 144-151, 1978.

[10] Lance Doherty, Kristofer S. J. Pister, and Laurent El Ghaoui. Convex position estimation in wireless sensor networks. *IEEE INFOCOM*, 2001.

[11] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. *17th Annual ACM Symposium on Principles of Distributed Computing*, pages 11-20, 1998.

[12] Lewis Girod and Deborah Estrin. Robust range estimation using acoustic and multimodal sensing. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[13] L.S. Heath. Graph embeddings and simplicial maps. *Theory of Computer Systems*, **30**:599-625, 1997.

[14] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *Hawaii International Conference on Systems Sciences*, 2000.

[15] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. *MOBICOM*, pages 56–67, 2000.

[16] David B. Johnson. Routing in ad hoc networks of mobile hosts. *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.

[17] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, **353**:153–181. Kluwer Academic Publishers, 1996.

[18] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. *International Conference on Mobile Computing and Networking*, pages 243–254, 2000.

[19] R. Koch, F.T. Leighton, B. Maggs, S. Rao, and A.L. Rosenberg. Work-preserving emulations of fixed-connection networks. *Journal of the ACM*, **44**:104-147, 1997.

[20] S.R. Kosaraju and M.J. Atallah. Optimal simulations between mesh-connected arrays of processors. *Journal of the ACM*, **35**:635-650, 1988.

[21] Sam Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a tiny aggregation service for ad hoc sensor networks. *Symposium on Operating Systems Design and Implementation*, 2002.

[22] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. *IEEE INFOCOM*, 1997.

[23] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, **36**(3):510-530, 1989.

[24] David Peleg. Informative labeling schemes for graphs. *Mathematical Foundations of Computer Science*, pages 579-588, 2000.

[25] David Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33:167-176, 2000.

[26] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *ACM SIGCOMM Conference*, 1994.

[27] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. *Military Communications Conference.*, 1997.

[28] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The Cricket location-support system. *International Conference on Mobile Computing and Networking*, 2000.

[29] Ananth Rao, Christos Papadimitriou, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Geographic routing without location information. *Mobicom*, 2003.

[30] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. *IEEE INFOCOM*, 2002.

[31] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. GHT: a geographic hash table for data-centric storage. *WSNA*, 2002.

[32] Arnold L. Rosenberg and Lenwood S. Heath. *Graph separators, with applications.* Kluwer Academic/Plenum Publishers, 2000.

[33] Antony Rowstron and Peter Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms* (Heidelberg, Germany, 12–16 November 2001), pages 329–350, 2001.

[34] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. Data-centric storage in sensornets. *HOTNETS*, 2002.

[35] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Conference* (San Diego, CA, 27–31 August 2001). Published as *Computer Communication Review*, **31**(4):149–160, 2001.

[36] Mikkel Thorup and Uri Zwick. Compact routing schemes. *SPAA*, 2001.

[37] Fan Ye, Alvin Chen, Songwu Lu, and Lixia Zhang. A scalable solution to minimum cost forwarding in large sensor networks. *ICCCN*, 2001.

[38] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. *Tapestry: an infrastructure for fault-tolerant wide-area location and routing.* UCB Technical Report UCB/CSD–01–1141. Computer Science Division (EECS) University of California, Berkeley, April 2001.