Practical 3D Geographic Routing for Wireless Sensor Networks

Jiangwei Zhou
The School of Electronic and Information Engineering
Xi'an Jiaotong Univeristy
zjw17@stu.xjtu.edu.cn

Yu Chen
Department of Computer Science
Duke University
yuchen@cs.duke.edu

Ben Leong, Pratibha Sundar Sundaramoorthy
Department of Computer Science
National University of Singapore
benleong@comp.nus.edu.sg, psundars@gmail.com

Abstract

Geographic routing is of interest for sensor networks because a point-to-point primitive is an important building block for data-centric applications. While there is a significant body of work on geographic routing algorithms for two-dimensional (2D) networks, geographic routing for practical three-dimensional (3D) sensor networks is relatively unexplored. We show that existing 2D geographic routing algorithms like CLDP/GPSR and GDSTR perform poorly in practical 3D sensor network deployments and describe GDSTR-3D, a new 3D geographic routing algorithm that uses 2-hop neighbor information in greedy forwarding and 2D convex hulls to aggregate node location information. We compare GDSTR-3D to existing algorithms, including CLDP/GPSR, GDSTR, AODV, VRR and S4, both in a real wireless sensor testbed and with TOSSIM simulations to show that GDSTR-3D is highly scalable, requires only a modest amount of storage and achieves routing stretch close to 1.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*

General Terms

Algorithms,Performance

Keywords

3D geographic routing, sensor networks, GDSTR

1 Introduction

There is a significant body of work on geographic routing algorithms for two-dimensional (2D) networks [1, 10, 15, 17, 18]. Geographic routing is of interest for sensor networks because a point-to-point primitive is an important building block for data-centric applications [8, 21, 25]. They are typically more scalable than alternatives because by exploiting geometric information on the network topology, they can guarantee packet delivery with less storage [10]. Typically, the storage required is proportional to node density, and this makes them practical for implementation in sensor operating systems like TinyOS, which does not support dynamic memory allocation.

More recently, there have been a number of practical deployments of three-dimensional (3D) sensor networks [3, 7, 9], and also a corresponding interest in geographic routing algorithms for 3D networks [4, 5, 19]. However, to the best of our knowledge, the 3D geographic routing algorithms proposed are mainly only of theoretical interest because they are designed to work with special topologies like *unit ball graphs* (UBGs).

In this paper, we show that existing 2D geographic routing algorithms like CLDP/GPSR [10, 11] and GDSTR [17] perform poorly in practical 3D sensor network deployments and build upon GDSTR to develop a 3D geographic routing algorithm that works well in a practical 3D sensor network deployment. We highlight two key observations from our work.

One, geographic routing in 3D topologies is intrinsically harder than routing in 2D topologies [4], since greedy forwarding tends to encounter more local minima in general 3D topologies. Surprisingly, we found that the simple strategy of extending greedy forwarding by using 2-hop neighbor information significantly improves the greedy forwarding success rate for 3D networks and thereby improves geographic routing performance.

Two, it is not entirely straightforward to extend existing 2D geographic routing algorithms to implement a 3D geographic routing algorithm because three-dimensional data structures are complicated to implement and costly to both store and compute. Among the previously proposed 2D algorithms, GDSTR [17] which uses 2D convex hulls is a natural candidate for extension to three dimensions because it is theoretically oblivious of dimensionality. However, we found the naive approach of replacing 2D convex hulls with 3D convex hulls to be impractical. 3D convex hulls require significantly more storage and are much more computationally costly, since in addition to the points on the hull, auxiliary

SenSys'10, November 3–5, 2010, Zurich, Switzerland.

data structures are needed to support operations with them. Also, the CC2420 radio used by many sensor nodes supports only packets up to 127 bytes in size and the cost of transmitting 3D hulls can be prohibitive.

We found that we can extend GDSTR to three dimensions by approximating 3D convex hulls with two 2D hulls and still achieve excellent routing performance for 3D networks. Our new algorithm, which we call *GDSTR-3D*, forwards packets greedily as long as it can find a neighbor closer to the destination than the current node. If the packet ends up in a local minimum, the node then attempts to forward the packet to a neighbor that has a neighbor closer to the destination than itself. If this fails, GDSTR-3D switches to forwarding the packet along the edges of a spanning tree which aggregates the location of the nodes in its subtrees using two 2D convex hulls.

We show in both testbed experiments and TOSSIM simulations that GDSTR-3D is able to guarantee packet delivery and achieve hop stretch close to 1. GDSTR-3D also scales well for networks of up to 3,200 nodes, and has smaller memory and maintenance bandwidth requirements compared to other point-to-point routing algorithms like AODV, VRR and S4 for large networks. Given these properties, we believe GDSTR-3D is an attractive choice for routing in large 3D wireless sensor networks. For small sensor networks, any of the existing routing algorithms would probably suffice.

The remainder of this paper is organized as follows: in Section 2, we provide a review of existing and related work. In Section 3, we describe GDSTR-3D and its implementation in TinyOS. We present our experimental results on the Indriya wireless sensor testbed [3] in Section 4 and our simulation results on TOSSIM in Section 5. Finally, we discuss our work in Section 6 and conclude in Section 7.

2 Related Work

Geographic routing algorithms were first proposed for ad hoc wireless networks because it was believed that they would be able to guarantee the delivery of packets using only local information [1, 10]. The original proposals used two-dimensional coordinates and are based on *face routing* [14] (originally called Compass Routing II), including GFG [1], GPSR [10], GPVFR [18] and the GOAFR+ family of algorithms [15]. The correctness of these algorithms depended on certain assumptions on the network graph, which were unfortunately typically violated in practical networks, rendering them infeasible in practice [26].

Kim et al. developed the Cross-Link Detection Protocol (CLDP) [11, 13] which can planarize arbitrary network graphs correctly. Unfortunately, it turns out that planarization is an extremely costly process [17]. A lazy variant of CLDP that only performs the planarization on-demand helps in some scenarios [13], but it does not change the fact that planarization is still costly when required, especially for three-dimensional graphs. Also, it does not help that face routing algorithms are sensitive to the face-change rule [12]. An efficient face-change rule does not guarantee correctness, while a correct face-change rule typically requires the complete exploration of a face and is therefore extremely costly. Face routing algorithms are also typically not very efficient at finding the shortest path around a routing void because they do not have access to sufficient information to decide on the direction of void traversal [18].

Leong et al. developed GDSTR [17] to address the issues with both face routing and the costs of CLDP planarization. The key idea is to use two distributed spanning trees rooted at opposite ends of the network to "approximate" a planar graph. When greedy forwarding fails, GDSTR provides delivery guarantees by effectively performing a depth-first search (DFS) on these trees. The search is highly efficient because each node in the tree aggregates information on all its children using a convex hull, thereby allowing the search tree to be pruned efficiently. While one tree is sufficient for correctness, the use of two trees provides GDSTR with two alternative paths when it needs to route around a void and sufficient hints to pick the right path. GDSTR was shown to perform better than other existing face routing algorithms like GPSR and GOAFR+ for 2D networks [17]. In this paper, we extend GDSTR to three dimensions using two 2D convex hulls to aggregate node locations, instead of using 3D hulls.

There has also been other recent attempts to extend 2D geographic routing algorithms to three-dimensional networks. Durocher et al. [4] proved that there does not exist a deterministic local routing algorithm for 3D networks that guarantees delivery of messages. To address this issue, Flury and Wattenhofer proposed a randomized geographic routing algorithm called *Greedy Random Greedy* (GRG) routing [5] for UBG networks that is based on random walk. They showed that if d is the length of the optimal path between a given pair of source and destination nodes, then the expected length of the route obtained by any randomized or even deterministic routing algorithm is given by $\mathtt{W}(d^3)$. While a random walk might be of theoretical interest, it is clearly not practical. In fact, their results show that GRG incurs message overhead greater than flooding for sparse networks between 2,000 and 5,000 nodes in size. Liu et al. proposed a three-dimensional analog to face routing called Greedy-Hull-Greedy (GHG) routing [19], based on network hulls constructed with partial unit Delaunay triangulations. Like planarization, the distributed computation of Delaunay triangulations is a hard problem [16] and so GHG is not likely to be usable in practical networks with arbitrary topologies.

Closely related to our work are geographic routing algorithms based on non-Euclidean coordinate systems, including VPCR [21], LCR [23], BVR [6], and S4 [20]. VPCR is based on polar coordinates, is not as efficient as geographic routing algorithms and incurs significant overhead under node and network dynamics. The others use coordinates based on landmark nodes (beacons). The major drawback is that they typically need to maintain a large number of beacons (about 30 to 40) to achieve routing performance comparable to geographic routing algorithms and they either have to resort to flooding when packets end up at local minima [6] or maintain $O(\sqrt{n})$ state [20], where n is the network size. If the addressing is based on hop counts to beacons, i.e. BVR, it is a practical concern for existing sensor networks because the IEEE 802.15.4-compliant CC2420 radio

hardware buffer is only 127 bytes in size and the addressing would take up most of the packet. S4 assumes the availability of a location service to map destination nodes to the corresponding beacon node.

Caesar et al. developed VRR [2], a routing algorithm based on distributed hash tables. To route to its successors on the virtual ring, a node sets up and maintains forwarding entries to its successors and predecessors along multi-hop physical paths. As a result, a node has routing table entries for paths towards its neighbors in the ring and also for the nodes on the paths between them. VRR greedily forwards a packet towards the node with the closest ID to the destination ID in the routing table. The routing state per node is roughly $O(\sqrt{n})$. For a large network, $O(\sqrt{n})$ can be quite significant and there are concerns over the message overhead required to maintain the state.

3 GDSTR-3D

There are many challenges in extending and implementing GDSTR for 3D sensor networks. First, because TinyOS does not support dynamic memory allocation, we need to pre-allocate memory for all the routing state that need to be stored per node. Second, the CC2420 radio supports only packets up to 127 bytes in size and has a limited data rate. Thus, it is not practical to transmit 3D convex hulls with a large number of vertices. Third, existing sensor motes like the TelosB motes [27] have limited DRAM and flash memory. Finally, the precision of floating point operations is limited, which might lead to inaccuracies in the numerical computations.

GDSTR forwards packet greedily until a local minimum is encountered. When a packet ends up in a local minimum, GDSTR uses *hull trees* to route the packet around the void in a deterministic way. A *hull tree* is a spanning tree where each node has an associated convex hull that contains within it the locations of all its descendant nodes in the subtree rooted at the node. GDSTR switches back to greedy forwarding once it finds a neighbor closer to the destination than the local minimum.

GDSTR-3D differs from GDSTR as described in [17] in the following ways:

1. *Aggregation with two 2D convex hulls instead of one.* In GDSTR, a 2D convex hull is used to aggregate the locations of the nodes in a subtree; in GDSTR-3D, two 2D convex hulls are used to approximate a 3D convex hull.
2. *Two-hop greedy forwarding.* GDSTR forwards packets greedily like other 2D geographic routing algorithms [1, 10, 15, 18]. GDSTR-3D uses two-hop information for greedy forwarding. While this requires slightly more storage in the nodes, it does not incur much additional maintenance overhead since existing sensor nodes already have to broadcast their neighbor information for bidirectional connectivity checks.
3. *Limit on hull tree fanout*. In GDSTR, the number of child nodes in the hull trees is not limited. Because TinyOS does not support dynamic memory allocation, the maximum fanout for GDSTR-3D has to be fixed at compile time. This change might increase the depth of the hull trees in some situations. In theory, such a limit can also cause some nodes to be disconnected, but with a sufficiently dense network, such a scenario is unlikely to arise in practice when the maximum fanout is sufficiently large.
4. *Conflict hulls not implemented.* In addition to its convex hull, each node in GDSTR also maintains information about the set of convex hulls, known as *conflict hulls*, that intersect with its own convex hull. GDSTR-3D does not implement this *conflict hull* optimization to reduce storage costs. The key drawback of not implementing the conflict hull optimization is that undeliverable packets would always be forwarded to the root, thereby increasing the load on the root nodes. We do not expect to have a significant number of undeliverable packets in practice. Furthermore, there is a simple optimization to address the overhead from undeliverable packets, which is for the root to send a packet to the source and the intermediate nodes to inform them that a given destination is undeliverable so that the undeliverable packets are dropped. In other words, this should not be an important concern in practice.

3.1 Routing Algorithm

In this section, we provide a detailed description of the GDSTR-3D routing algorithm.

GDSTR-3D first attempts to forward packets greedily, i.e. to the neighbor whose coordinates is strictly closer to the destination node in Euclidean distance than the current minimum. The *current minimum* is defined as the node among all the nodes previously visited by a packet that is closest to the destination, i.e. we will set the current node as the *current minimum* of a packet if it is closer to the destination than the existing current minimum. In GDSTR-3D, a node also records the one-hop neighbor information broadcast by its neighbors, so if none of its immediate neighbors is strictly closer to the destination, it will attempt to forward the packet instead to the neighbor that has a one-hop neighbor that is closer to the destination than the current minimum. Tree forwarding is guided by the convex hull information in the nodes and is guaranteed to find the destination if it is reachable. GDSTR-3D switches back to greedy when it finds a neighbor that is strictly closer to the destination than the current minimum.

The key insight in GDSTR (and GDSTR-3D) is that the convex hulls of the nodes uniquely define a *routing subtree* that must contain the destination node, if the packet is deliverable. The routing subtree is defined as the subtree comprising of all the nodes in the network whose hulls contain the coordinates of the destination node. If a packet is not deliverable, the routing subtree will be a null tree.

Tree Traversal. When a packet ends up in a local minimum, it switches to tree forwarding mode by picking a suitable routing tree between the two available trees. It has been shown that by picking the tree with a root nearer to the destination, GDSTR is typically able to pick the more efficient way to route around a void and achieve good routing performance [17].

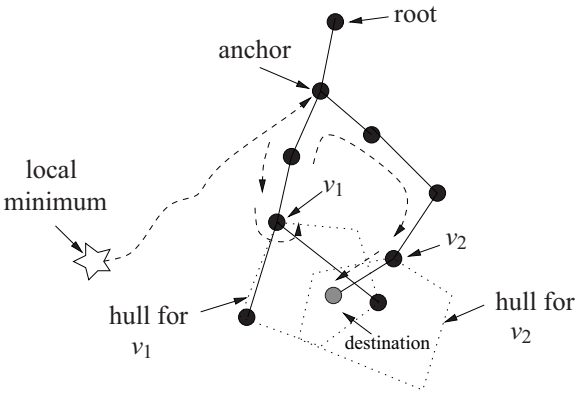


Figure 1. Overview of GDSTR tree traversal.

The packet is first forwarded up the tree until it reaches the routing subtree. Because the routing subtree must contain the root of the tree by construction, the packet will either reach a node on the routing subtree, or reach the root and is found to be undeliverable. Once the packet reaches a node on the routing subtree, we record this node as the *anchor node*. Tree traversal is simply a systematic traversal of the routing subtree using a right-hand rule, as illustrated in Figure 1.

Sketch of Correctness. Because the distance of the current minimum to the destination is always strictly decreasing in greedy forwarding, greedy forwarding is guaranteed to be loop-free. By construction, if a packet is deliverable, the destination node must be a leaf on the routing subtree and so if greedy forwarding is unable to deliver the packet, the delivery can be guaranteed by traversing the entire routing subtree systematically.

Packet State. Each GDSTR-3D data packet contains the following state information:

- *mode*: current forwarding mode (Greedy/Tree),
- n_{min}: the node that is the *current minimum*,
- n_{anchor}: the anchor node of the tree traversal, and
- *root*: tree identifier for the chosen forwarding tree.

mode indicates the currently used forwarding mode for the packet, namely greedy or tree. n_{min} is the node at which the packet switches from greedy to tree mode. It is used to determine when to switch back to greedy forwarding. n_{anchor} is the first node on the routing subtree encountered by the packet in tree forwarding mode. It is used to determine whether a packet is deliverable. *root* is the unique identifier of the tree which is currently used by the packet in tree forwarding mode.

Formal Description. The following is a detailed description of the GDSTR-3D routing algorithm. In the hull trees, each intermediate node maintains a fixed ordering of its child nodes.

Algorithm(GDSTR-3D). When a node v receives a packet p for destination t from a neighbor node u:

1. **Termination condition:**
   - v is the destination t, the packet is delivered.
   - p is *undeliverable*, if:
     - v is the root of the hull tree and its hull does not contain the destination t; or
     - $p.anchor = v$ and:
       - v is the root of hull tree and u is the last child node with a hull containing t; or
       - u is the parent of v.
2. **Check for switch to Greedy mode:** If $p.mode = Tree$ and there is at least one 1-hop or 2-hop neighbor whose distance to t is strictly less than $p.n_{min}$, then:
   - Set $p.mode$ to *Greedy*.
   - Reset $p.n_{anchor}$ and $p.root$ to NULL.

   If $p.mode$ is *Greedy* go to step 3, otherwise go to step 5.
3. **Greedy mode:**
   - Set $p.n_{min}$ to the current node v if v is strictly closer to the destination or if $p.n_{min}$ is NULL.
   - If there exists a 1-hop or 2-hop neighbor w that is strictly closer to t than $p.n_{min}$, then forward p to w if w is a 1-hop neighbor, or forward p to a 1-hop neighbor connected to w.
4. **Switch to Tree Forwarding mode:** Packet has reached a local minimum. Set $p.mode$ to *Tree*.
5. **Set tree information on p:** If $p.root$ is set, continue to step 7. Otherwise, set $p.root$ to the root of the tree closest to t and go to step 6.
6. **Find routing subtree and set anchor:** If $p.anchor$ is set, continue to step 7. Otherwise,
   - if v is a leaf node forward p to its parent in the tree with root $p.root$;
   - if v's hull contains the destination t, set $p.n_{anchor}$ to v and forward p to the first child whose hull contains destination t;
   - Else forward p to v's parent node in the tree with root $p.root$.
7. **Tree Forwarding:** v will choose the next hop based on its relationship with u.
   - If u is v's child:
     - Forward p to the parent if u is the last child of v that has a hull containing the destination t.
     - Otherwise, forward p to the next child after u that has a hull containing the destination.

- If u is v's parent, forward to the first child with a hull containing the destination if such a child exists. Otherwise, forward p back to u.

3.2 Hull Tree Construction

The construction of hull trees is done in two phases: tree building and convex hull aggregation.

In phase one, root nodes are elected for each hull tree. The root election process in GDSTR-3D is the same as that for the 2D version, i.e. pick the nodes with minimum and maximum x-coordinates (using the y and z coordinates to break ties). Each node broadcasts its view of the root and when a node learns of a new node with a smaller (or larger) x-coordinate, it updates its view and broadcasts the new information. Eventually all the nodes will come to a consensus on the two root nodes.

Each node picks the neighbor that has the minimal hop count to the root for the tree as its parent. Ties are broken by picking the node with the smaller distance to the root. In our implementation of GDSTR-3D, we bound the storage requirement by limiting the maximum number of children of each node to 5, with a simple handshake mechanism.

3.3 Aggregation of Node Coordinates

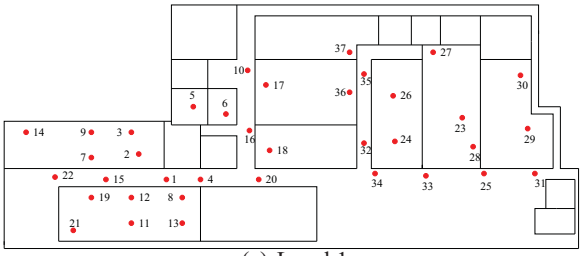
In phase two, the coordinates of the nodes in the tree are aggregated up the hull tree. Each node maintains information about the hulls of its child subtrees and transmits its hull, which is a region containing the locations of all the nodes in its subtree, to its parent node for each tree. Since GDSTR-3D uses two hull trees, each node will send separate aggregation unicast messages to its parent nodes for each tree. This message contains the identifier of the root node of the tree and the coordinates of the vertices of its hull. Where necessary, the size of the transmitted convex hull is reduced with the hull point reduction technique described in [17] to fit the hull into the message.

GDSTR uses 2D convex hulls. A straightforward extension to three dimensions would be to use 3D convex hulls. It turns out that the implementation of 3D hulls in TinyOS is not very practical. Auxiliary data structures like faces and edges are needed to support operations like aggregation and membership testing. This takes a lot of memory and incurs high computational complexity.

We observed that the hulls are used to aggregate the coordinates of the nodes in a subtree for quick membership testing. In this light, we found that it is sufficient to use 2D convex hulls to approximate a 3D hull by projecting nodes onto orthogonal 2D planes. This process can be done in the xy, xz and yz planes. We found that using 2D convex hulls in two of these three planes was sufficient to achieve good performance, so GDSTR-3D uses two 2D convex hulls in each hull tree. We also investigated the use of a sphere to perform the aggregation, but we found that a sphere does not work well for sparse networks.

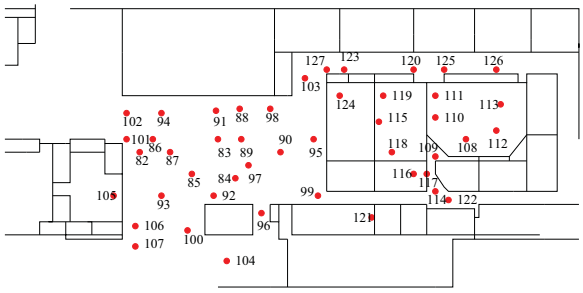
4 Evaluation on 3D Sensor Network Testbed

We evaluated GDSTR-3D by comparing it to AODV [22], CLDP/GPSR [10, 11], GDSTR [17], S4 [20] and VRR [2] on Indriya [3], a wireless sensor network testbed deployed at the National University of Singapore. On Indriya, we had



(a) Level 1

(b) Level 2



(c) Level 3

Figure 2. Node locations on different floors of the Indriya testbed.

access to 120 TelosB [27] sensor motes deployed on three levels of a building. The node locations for the testbed are shown in Figure 2. Each mote has a TI MSP430 processor running at 8 MHz, 10 KB of RAM, internal and external flash memories of size 48 KB and 1 MB respectively, and a Chipcon CC2420 radio operating at 2.4 GHz with an indoor range of approximately 20 to 30 meters.

Since we implemented GDSTR-3D, it was relatively easy to modify it to obtain GDSTR (without conflict hulls). The source codes for CLDP/GPSR, S4 and VRR were obtained from the respective authors. The implementation of AODV used was taken from the source code distribution of VRR [2]. Because the sources were in TinyOS v1.x, we had to port them to TinyOS v2.x. Care was however taken to ensure that bugs were not introduced in the porting and in the process, some minor bugs in AODV and VRR were identified and fixed.

For S4, we used $\sqrt{n}$ beacons, where n is the network size, because this achieves a good balance between the storage cost of the routing state and the performance according to Mao et al. [20]. We also enable the scoped distance vec-

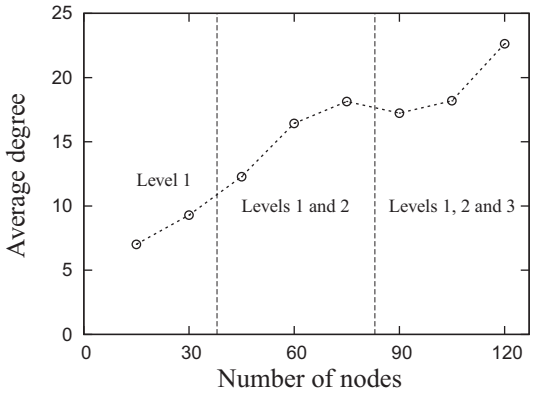


Figure 3. Plot of average node degree against network size on Indriya.

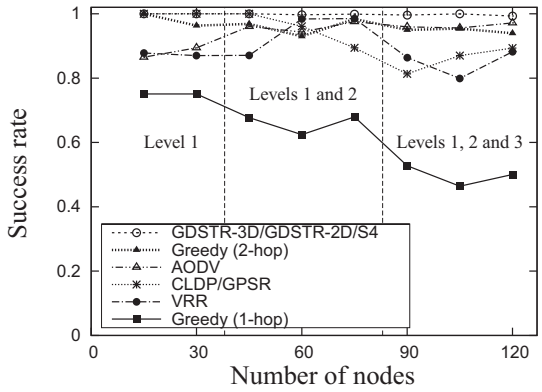


Figure 4. Plot of packet delivery success rates for various algorithms on Indriya.

tor (SDV) and reliable broadcast. For VRR, AODV and CLDP/GPSR, we used the default settings in the code by the authors. For GDSTR, we used two trees and limited the maximum number of children for each node to 5.

To evaluate the performance of these algorithms, we ran experiments with connected subsets of nodes. First, we started with only sets of randomly-chosen nodes on Level 1; next, we used randomly-chosen nodes on both Levels 1 and 2; finally, we used randomly-chosen nodes distributed across all three levels. Experiments were repeated with 5 random subsets of nodes for each network size. The average densities of the resulting network topologies are shown in Figure 3. The average node degree falls slightly for 90 nodes because it turns out that the network topologies are less dense when 90 nodes are distributed over three floors, compared to 75 nodes distributed over two floors.

For the geographic routing algorithms, GDSTR-3D, GDSTR, and CLDP/GPSR, coordinates were assigned to the nodes based on their actual physical locations. Because GDSTR and CLDP/GPSR are 2D algorithms, they ignore the z (height) coordinate and work only with the x and y coordinates.

In each experiment, we booted up the nodes sequentially and let the algorithms converge for 15 minutes, which we found was more than sufficient for the tested algorithms, except VRR, for which we had to wait for 50 minutes. Next, we routed packets between randomly chosen source and destination pairs for another 45 minutes. For VRR, since the convergence is significantly slower, we let it run for 10 minutes. For S4 and GDSTR, we sent one data packet every 200 ms. For VRR, we sent one data packet per second to use the same parameter settings as the original paper [2]. For CLDP/GPSR, we also sent one data packet per second since it usually incurs a larger hop stretch and each packet stayed in the network for a longer time, thereby increasing the probability of collisions.

AODV was evaluated somewhat differently because it is a reactive protocol. Our current implementation of AODV is also naive; when a node receives a packet for which it does not have a path cached, the packet is dropped and a route request is sent. So, for the first 15 minutes, we randomly selected source and destination pairs using a fixed random seed and routed the packets between those pairs at a rate of one packet every second to fill the route caches. For the next 45 minutes, we used the same random seed and routed packets between the same pseudo-randomly picked source and destination pairs to measure routing performance.

4.1 Success Rate

In Figure 4, we plot the packet delivery rates for the various algorithms. These numbers exclude link-level losses, which are inevitable in a practical setting.

We observed that as the nodes are spread across more levels, the greedy forwarding success rate drops steeply from 90+% to about 40%. This fits with our intuition that routing in three dimensions is harder than routing in two dimensions. It was however quite surprising that by simply using 2-hop neighbor information, we can achieve a greedy forwarding success rate of approximately 95% even when nodes from all three levels are used.

As expected, GDSTR-3D, GDSTR and S4 are able to achieve 100% packet delivery success rates. While CLDP achieves a delivery success rate of 100% when all nodes are in the same level of the building, delivery failures are experienced when the nodes are spread across two or more levels. These losses are likely caused by the faulty first-intersection face change rule used by CLDP/GPSR [12]. The success rate of AODV is also close to 100%, because we routed the packets during the first 15 minutes to fill the route caches.

We found that the success rate of VRR is less than 100% in our experiments because the link failure detection protocol implemented does not always work. In particular, in the VRR implementation, a link is disabled if the percentage of received packet falls below a threshold, and re-enabled when the reception ratio is greater than it, using an exponentially weighted moving average (EWMA) filter. However, our testbed has a fair number of lossy links, where the packet losses vary over time. Link quality oscillations might then trigger the frequent setup and tearing down of paths, causing packet losses.

Also, VRR seems to be highly sensitive to its configuration parameters. In our evaluations, we used the default parameters that came with the VRR source code. It was not clear how the parameters ought to be tuned. The parameters include (i) the join interval, which specifies how often a node

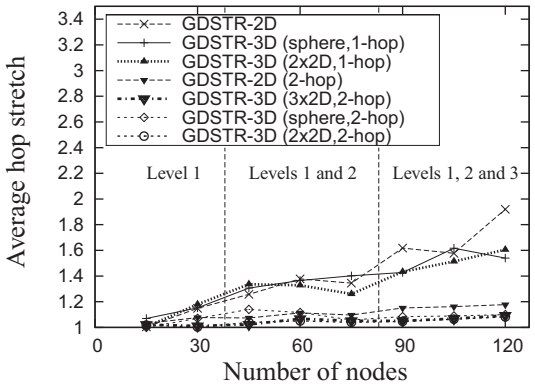


Figure 5. Plot of hop stretch for GDSTR with different settings on Indriya.

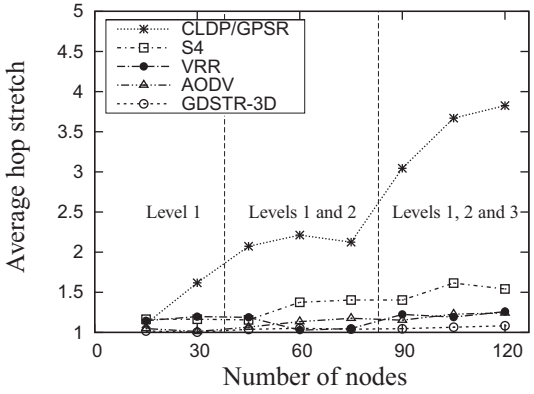


Figure 6. Plot of hop stretch for various algorithms on Indriya.

will try to set up virtual paths, (ii) the probe timeout, which specifies how often a path will expire if a node is waiting for repair, (iii) the first join interval, which specifies how long we should wait before the link state stabilizes and (iv) some other parameters for the link quality estimation. In the original VRR paper [2], the success rate was reported to be 100% for 67 nodes. In our experiments, VRR also achieves approximately 100% success for networks with 70 nodes, but it does not perform quite as well when the number of nodes is significantly larger or smaller.

4.2 Hop Stretch

We define *hop stretch* as the ratio of number of hops in the path between two nodes achieved by a routing algorithm to the number of hops in the shortest path between them. This is a measure of routing efficiency.

First, we investigated how the hop stretch varied for different variants of GDSTR. We investigated the effect of three factors: (i) 2D versus 3D; (ii) different coordinate aggregation methods; and (iii) 1-hop greedy forwarding versus 2-hop greedy forwarding. For coordinate aggregation, we compared the use of a sphere to 2×2D convex hulls and 3×2D convex hulls. The results are shown in Figure 5.

As expected, GDSTR-2D performs worst, even though it guarantees packet delivery. We applied GDSTR-2D by projecting the coordinates onto the *xy*-plane, and this means that greedy forwarding is unable to exploit height information. Two nodes that are close in the projection on the *xy*-plane may not be spatially close to each other in 3D networks. What is surprising however is that GDSTR-3D with one-hop greedy forwarding is not significantly better. This is perhaps best explained by the greedy forwarding success rates in Figure 4. One-hop greedy forwarding has a high likelihood of ending up in a local minimum. On the other hand, GDSTR-2D with two-hop greedy forwarding improves hop stretch significantly for larger network sizes.

The best results were obtained for GDSTR-3D with two-hop greedy forwarding. The aggregation method seems to have very little effect since two-hop greedy forwarding already has a high success rate. However, we show in the next section that the aggregation method matters for low density networks.

In Figure 6, we compare the routing performance of GDSTR to CLDP/GPSR, S4, VRR and AODV. It is clear from these results that CLDP/GPSR is ill-suited for routing in 3D, though incorporating 2-hop greedy forwarding can likely improve its performance significantly for Indriya. S4 generally performs quite well; however, since the beacon selection is random and the network is small, routing first to the beacons might result in one or two additional hops, thereby leading to a larger than optimal stretch.

The hop stretch of GDSTR-3D is very close to 1. AODV also achieves a hop stretch close to 1 for small networks but the stretch increases for larger networks. Because AODV maintains routing tables by route discovery and updates the table when a better route is discovered, it is not surprising that the hop stretch is good since the Indriya is relatively small and the maximal diameter is 5 hops.

The hop stretch of VRR is interesting. It is close to 1 for networks with 60 and 75 nodes. However, like what we observed in Section 4.1 for success rates, hop stretch is higher in networks that are significantly larger or smaller. We suspect that this has something to do with the tuning of the parameters for VRR but we have not been able to verify this hypothesis.

4.3 Size of Compiled Binaries & Source Code

The sizes of the compiled binaries and the lines of code for the implementations of the algorithms studied are shown in Table 1. The size of the compiled binaries is an important consideration for a practical sensor network because existing sensor motes have limited memory. For example, the TelosB mote [27] has only 48 KB of flash memory for the executable binary. From these figures, it is clear that we do not have a lot of memory left to add more features for most of the algorithms. S4 assumes that availability of a location service. Given that the S4 binary takes up 43 KB out of 48 KB of the executable memory, it would be a challenge to fit a location service in the remaining 5 KB of memory.

The number of lines of code required to implement each algorithm, while not exactly an accurate or robust metric, provides us with an indication of the relative complexities of the algorithms investigated.

While reactive algorithms like AODV have typically been regarded to be unscalable, the figures in Table 1 suggests that for small sensor networks with cache-friendly traffic patterns

Table 1. Relative sizes of algorithms investigated.

Algorithm	Compiled Binary Size (KB)	Lines of Code
GDSTR-3D	39.5	2,757
GDSTR [17]	33.8	2,641
CLDP/GPSR [10, 11]	47.5	2,500
S4 [20]	43.2	3,997
VRR [2]	45.1	4,135
AODV [22]	21.1	1,294

and delay-insensitive applications, AODV can be an attractive option, given its relative simplicity compared to the other existing point-to-point routing algorithms.

5 Simulation Results

While our experiments on the Indriya sensor network testbed provides us with insights on the performance of geographic routing in a practical setting, testbed experiments have limitations. The most obvious limitation is that of scale. Indriya has only 120 nodes. Few testbeds have more than several hundred nodes.

Moreover, while we can obtain a range of network topologies by selecting subsets of the available nodes to work with, it is difficult for us to adjust the density of the network to study its impact on geographic routing. In this light, we conducted a set of simulation experiments on TOSSIM, a simulator for TinyOS, in order to study (i) the effect of network density, and (ii) the performance and costs of various algorithms as we scale up to larger networks. In our simulations, we used an ideal radio channel and consider only application-level losses and ignore all link-level (collision) losses. We ignored the link losses since our evaluations in the Indriya testbed already incorporates the effects of link losses and our goal in this section is to compare the basic algorithmic behavior of GDSTR-3D to other routing algorithms.

Since CLDP/GPSR performs poorly in terms of routing stretch and packet delivery success rate and it incurs high message overhead due to its periodic probing, we take GDSTR (2D) [17] as the representative 2D geographic routing algorithm in our simulation experiments. We compare GDSTR-3D to GDSTR-2D, AODV [22], S4 [20], and VRR [2].

In addition to packet delivery success rates and routing performance, we also attempt to quantify the costs of the various algorithms in terms of storage and message overhead. The error bars shown for the plots in this section are for 95% confidence intervals.

5.1 Effect of Network Density

To study the effect of network density, we generated 200-node connected random topologies within a fixed 3D cubic space of $2,000 \times 2,000 \times 2,000$ and varied the network density by varying the radio range. Fifty different networks were generated for each density.

We used a simple radio model: we choose a fixed radio range for all nodes and two nodes can communicate if and only if they are within radio range. Nodes are added at random in the cubic space and we ensure that the topologies generated are connected by removing nodes that are not connected and repeating the process until the required number of nodes (200) is reached.

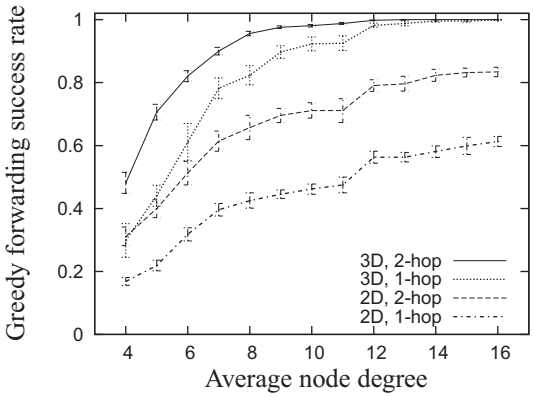


Figure 7. Plot of greedy forwarding success rate against network density.

In each experiment, we sent a packet between every possible pair of source and destination nodes, for a total of $200 \times 199 = 39,800$ packets for each experiment. For each simulation, we boot the nodes sequentially and sent one packet every 500 ms over a period of 20,000 s for each network.

Like in Section 4, AODV was evaluated somewhat differently from the other algorithms for the simulations because it is a reactive algorithm. While most algorithms were given 15 minutes to converge prior to the sending of the data packets, for AODV, packets were routed between source and destination pairs chosen with a random seed to fill the AODV route caches during this initial 15 minutes. The same random seed was subsequently used to generate the data packets to measure routing performance. We also do not plot the storage and overhead for AODV against the other proactive algorithms because it is not meaningful to do so.

5.1.1 Greedy Forwarding Success Rate

In Figure 7, we plot the greedy forwarding success rates for different network densities and for different forwarding strategies in the studied topologies. In particular, we plot the greedy forwarding success rates using 3D coordinates and using 2D projections onto the *xy*-plane, with local information of either one- or two-hop neighbors.

We can see that 3D greedy forwarding with 2-hop neighbor information achieves the highest greedy success rate, followed by 3D greedy forwarding with 1-hop neighbor information. There is a gap between these two strategies for average node degree below 12; above average node degree 12, 1-hop greedy forwarding is sufficient to achieve almost 100% forwarding success. From these results, it is also clear that with 2-hop greedy forwarding in 3D, the choice of geographic routing algorithm is not important for dense networks since greedy forwarding alone is often sufficient for packet delivery.

Another key observation is that even though forwarding success rates for 2D projections will improve with network density, greedy forwarding success seems to be capped asymptotically at approximately 80% even with 2-hop neighbor information. This suggests that 2D routing is not suffi-

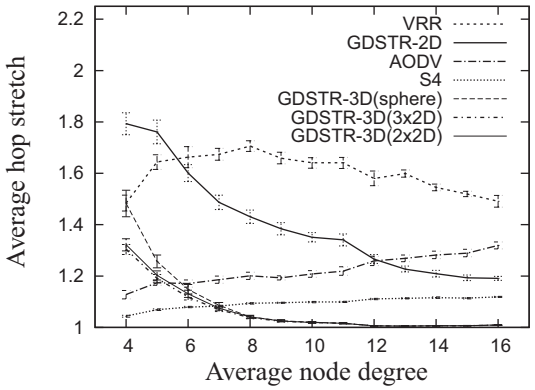


Figure 8. Plot of hop stretch against network density.

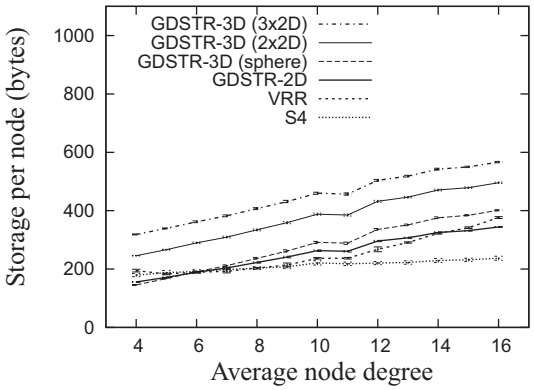


Figure 9. Plot of average storage requirement against network density.

cient to achieve good performance, even for dense 3D networks.

5.1.2 Routing Performance

Figure 8 compares the hop stretch achieved by various algorithms across different densities.

For GDSTR-3D, we also investigated the effect of aggregating coordinates using three different data structures: 2×2D convex hulls, 3×2D convex hulls, and a sphere. Our results show that the hop stretch of GDSTR-3D (2×2D) is almost the same as GDSTR-3D (3×2D). As expected, GDSTR-3D (sphere) has a higher hop stretch than GDSTR-3D using 2D convex hulls because a sphere aggregates the node locations less compactly than 2D convex hulls, and this results in a higher false positive rate during tree traversal. This effect is less pronounced for densities higher than 8 because of the high greedy success rates of around 95%. Because greedy forwarding success rates are significantly lower when forwarding using the 2D projection than that when using 3D coordinates, GDSTR-2D performs poorly for 3D networks.

The stretch for S4 is comparable to GDSTR-3D. In fact, S4 achieves better stretch for node densities below 7, while GDSTR-3D has better performance above this threshold. They show opposite trends: the hop stretch for GDSTR decreases with increasing density, while the hop stretch for S4 increases with increasing density. This is because GDSTR-3D benefits significantly from the higher greedy forwarding success rates at higher network densities. When the network density is higher, the optimal routing path is closer to a straight line; the beacons used by S4 are unlikely to lie on the line connecting the source and the destination and therefore S4 would incur some extra hops. For low density networks, greedy forwarding has a much higher likelihood of ending up in a local minimum and invoking the more expensive tree forwarding mode.

VRR performs quite poorly, and achieves a stretch around 1.6 (though it is somewhat independent of network density). We suspect that if the parameters for VRR were tuned appropriately, it might be possible to improve the routing performance, though we did not manage to determine how to better tune VRR despite trying many different parameter settings. AODV performs somewhat better and achieves stretch between 1.2 and 1.3.

5.1.3 Storage Cost

In Figure 9, we plot the average storage required per node against network density. The storage incurred by GDSTR consists mainly of the tree information, especially hull information of child nodes stored by each parent node. It is quite clear that among the variants of GDSTR, GDSTR-3D (3×2D) which stores three 2D convex hulls per node will require the most storage followed by GDSTR-3D (2×2D) and GDSTR-2D. GDSTR-3D (sphere) needs to store only the center and radius of its spherical hull and so incurs the least storage among the four. In general, the storage cost for GDSTR increases linearly as the network density increases since with increased densities, there are more neighbors.

S4 requires the least storage, mainly because it does not use floating point values. The storage for VRR is comparable to GDSTR-2D and the storage requirement also increases with network density. All the algorithms are relatively compact and on average requires less than 600 bytes of storage for 200-node networks for the range of network densities investigated.

It turns out that the storage requirement is not uniform among all the nodes and so we also plot the maximum required storage per node in Figure 10. In fact, the maximum storage required per node is the key factor that determines whether an algorithm can be deployed in a practical network. Note that the TelosB motes only have 10 KB of RAM. The maximum storage in VRR is about 2 to 6 times the average node storage, while the maximum storage for the rest is about 2 to 3 times the average storage. None of the algorithms requires more than 1,500 bytes of storage across the range of densities studied.

We did not include the storage of AODV in these figures, since the average storage required by AODV is dependent on the traffic pattern, and the maximum storage required is merely the size of the cache. For a 200-node network, storage is most definitely not a major concern. Even in the worst case where we allocate 200 entries for the cache, the total storage is only 1,000 bytes (since each entry is 5 bytes in size).

5.1.4 Control Traffic Overhead

We define the *message overhead* as the number of messages sent per node before an algorithm converges. We de-

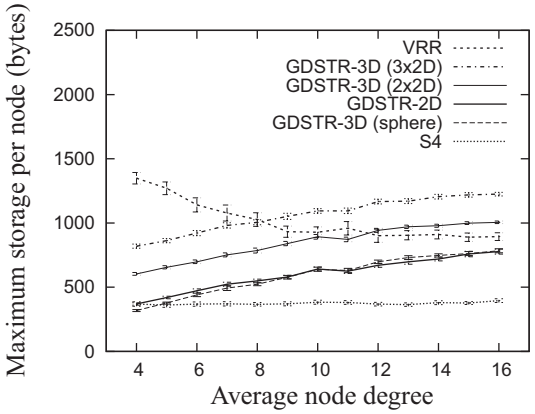


Figure 10. Plot of maximum storage requirement against network density.

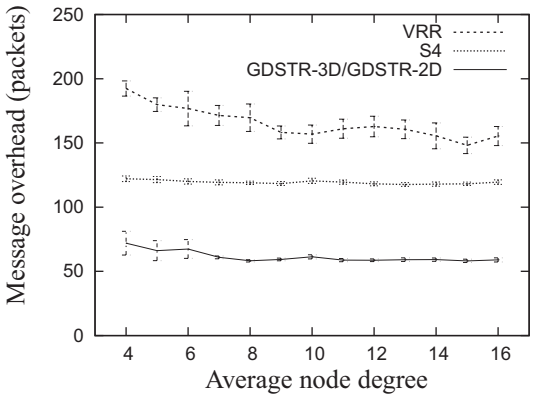


Figure 11. Plot of control traffic overhead required for convergence per node in number of packets against network density.

fine *convergence* as the state when all the nodes in the topology have gathered enough information required by the routing algorithm so that no further state updates are necessary. We measured message overhead in terms of the number of messages sent per node as shown in Figure 11 and the total bandwidth consumed per node for those messages as shown in Figure 12. Note that these numbers do not include the periodic stayalive messages where nodes broadcast their views of their known neighbors. Such messages are needed to allow the nodes to check for bidirectional connectivity and to react to node failures.

For GDSTR, the root information is propagated from the root node to the leaf nodes and hull information is aggregated from the leaves to the root. Hence, the number of messages sent per node is dependent mainly on the depth of the spanning tree formed, which depends on the diameter of the network and the position of the root node. There is therefore little difference in the number of messages that needs to be sent for stabilization between the different variants of GDSTR. Different hull aggregation methods will however affect the size of the messages and the difference in bandwidth requirements between the various GDSTR variant is shown in Figure 12.

Using 3×2D convex hulls requires almost twice as much bandwidth as spheres and GDSTR-2D. That said, the bandwidth overhead at 2,000+ bytes per node is relatively small. Since GDSTR-3D (2×2D) shows similar performance as GDSTR-3D (3×2D), with a lower storage requirement and message overhead, we will only compare GDSTR-3D (2×2D) with other algorithms in the subsequent sections.

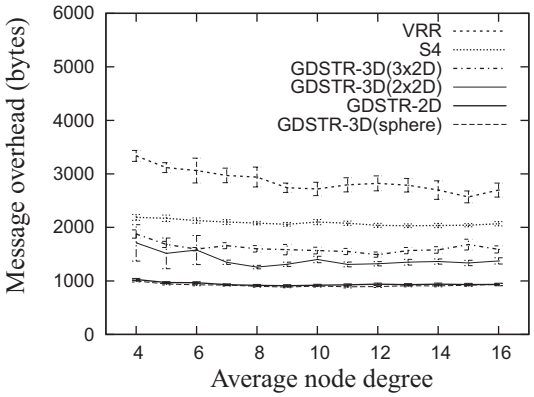


Figure 12. Plot of control traffic overhead required for convergence per node in bytes against network density.

The requirements for VRR and S4 are slightly higher than that for GDSTR-3D. AODV is a reactive protocol while other algorithms are proactive. The control overhead of AODV is dependent on the traffic pattern and the cache size, hence it is not meaningful to plot the control overhead of AODV for comparison with the other algorithms. Our key observation from these experiments is that the overhead for all the proactive algorithms studied is somewhat independent of network density.

5.2 Scaling Up

To investigate how various algorithms performed for large network sizes, we generated random topologies from 50 to 3,200 nodes in size. For the network sizes 200 and smaller, we used a $2,000 \times 2,000 \times 2,000$ cubic space, while for larger network sizes, we used a $3,000 \times 3,000 \times 3,000$ cubic space. By varying the radio range appropriately, we generated two sets of networks: low density networks with average node degree around 7 and high density networks with average node degree 16. For each network size, we ran the experiments on 50 random topologies. On each topology, we forward approximately 40,000 packets, where each node will send an equal number of packets to randomly-selected destinations. The source nodes send the packets in sequential order, i.e. no two source nodes will send packets at the same time. One data packet is sent every 500 ms over a period of 20,000 s for each network. As before, we compared GDSTR-3D to AODV, GDSTR-2D, VRR and S4.

5.2.1 Greedy Forwarding Success Rate

To characterize the network topologies studied, we plot the greedy forwarding success rates when using 2D/3D coordinates and either one- or two-hop neighbor information in Figures 13 and 14. For low density networks, it is clear that greedy forwarding becomes a harder problem as the network size scales up. Two-hop greedy forwarding helps somewhat, but is not sufficient to achieve 100% packet delivery. When the network density is sufficiently high (at average node degree 16), it is possible to achieve greedy forwarding success

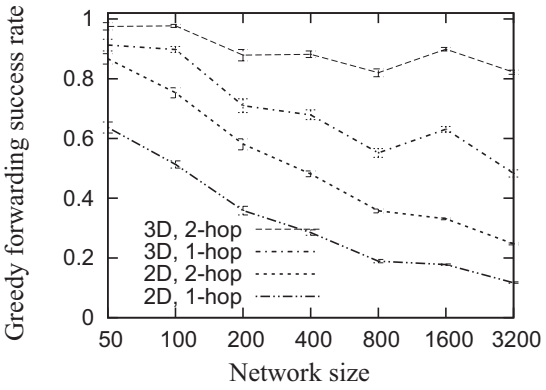


Figure 13. Plot of greedy forwarding success rate against network size for low-density networks (average node degree 7).

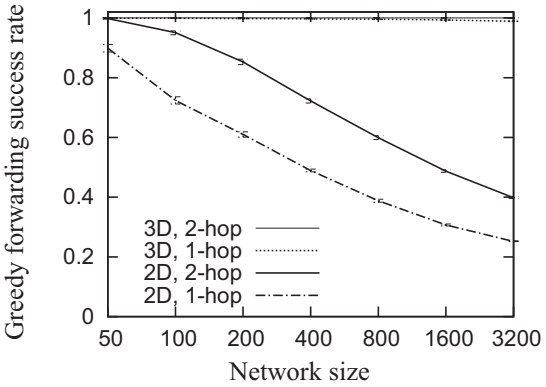


Figure 14. Plot of greedy forwarding success rate against network size for high-density networks (average node degree 16).

rates close to 1 with 3D greedy forwarding, which means that any 3D geographic routing algorithm will work very well. It is also clear from these results that the use of 2D coordinates (projections onto *xy*-plane) is definitely a bad idea for large networks independent of network density.

5.2.2 Routing Performance

In Figure 15, we compare the hop stretch for different routing algorithms with respect to network size for low density networks (average node degree 7). Under such a scenario, S4 achieves the best and uniformly good stretch as the network size scales up. GDSTR-3D achieves comparable results, though the stretch seems marginally higher. The performance gap between GDSTR-3D using 2-hop greedy forwarding and GDSTR-3D using only 1-hop greedy forwarding increases with network size. The stretch for 1-hop greedy forwarding remains under 1.5 even at 3,200 nodes. The stretch for GDSTR-2D even with 2-hop greedy forwarding is extremely high as the network size scales up. GDSTR-2D with 1-hop greedy forwarding achieves an even worse stretch, thereby demonstrating that it is indeed infeasible to apply 2D geographic routing algorithms on large 3D networks.

The stretch for VRR is significantly higher than that for GDSTR-3D and S4 and it gets worse with larger network

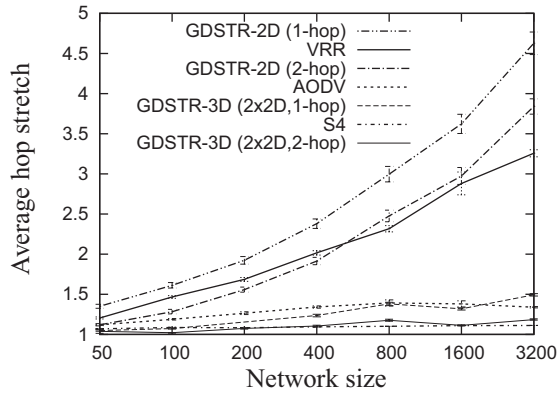


Figure 15. Plot of hop stretch against network size in low-density networks (average node degree 7).

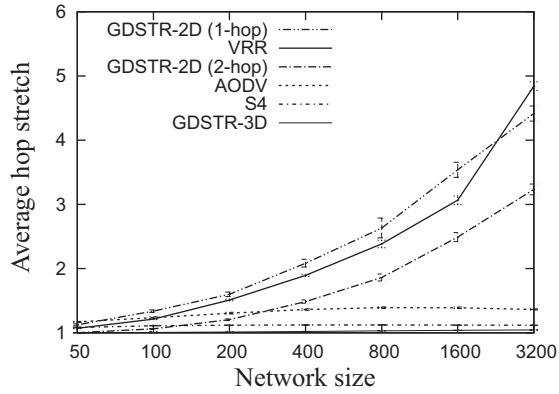


Figure 16. Plot of hop stretch against network size in high-density networks (average node degree 16).

sizes. When the network size increases, both the size of the VRR virtual ring and the number of hops in each virtual VRR link increases. Because it is quite unlikely that a virtual path will lie on the optimal path between a source and destination pair, the increase in the number of hops in the virtual links increases the stretch.

In Figure 16, we plot the hop stretch for different routing algorithms with respect to network size for high density networks (average node degree 16). There is only one line for GDSTR-3D because the performance for 1-hop and 2-hop greedy forwarding are indistinguishable (due to high greedy forwarding success rates). We find that GDSTR-3D achieves stretch very close to 1. S4 is marginally worse (though less than 1.2). The stretch for both GDSTR-3D and S4 remain uniformly low even as the network size scales up. In contrast, the stretch for GDSTR-2D and VRR get significantly worse as the network size scales up.

From Figures 15 and 16, we observe that AODV also achieves relatively good hop stretch independent of network size. The simulations for AODV for large networks however took significantly longer to complete than the other algorithms, most likely because of large amounts of traffic generated by the flooding of the route requests.

To address potential concerns that the generated unit ball graphs are not representative of real 3D sensor network topologies, we repeated this experiment with a set of topolo-

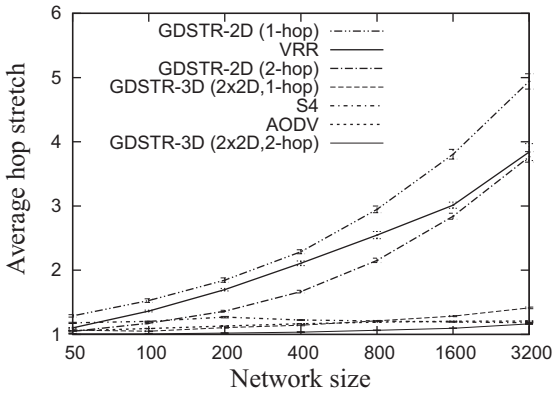


Figure 17. Plot of hop stretch against network size with multiple obstacles (average node degree 10).

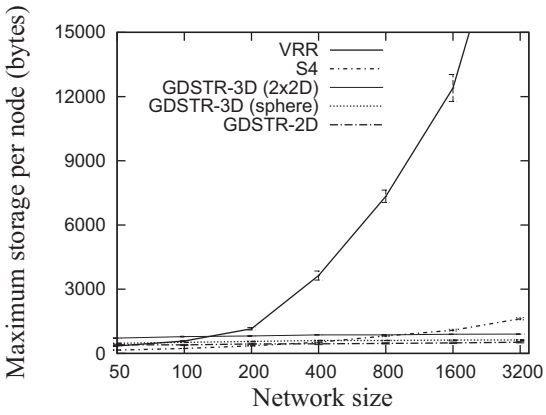


Figure 18. Plot of maximum storage requirement against network size in low-density networks (average node degree 7).

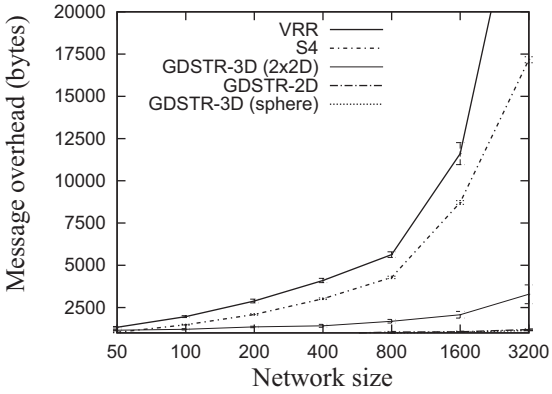


Figure 19. Plot of control traffic overhead required for convergence per node in bytes against network size in low-density networks (average node degree 7).

gies with obstacles. The networks with obstacles were generated by scattering at random approximately $\sqrt{\frac{n}{50}}$ cross-shaped obstacles in n-node topologies. Each cross consists of three perpendicular planar squares intersecting in the middle. The width of the crosses were $\frac{1}{4}$ of the region width for all network sizes. The effect of these obstacles is to cut off communication between nodes where the line-of-sight intersects an obstacle. For consistency, we varied the radio range of the nodes to ensure that the average node degree remains at 10 for the range of network sizes studied. As shown in Figure 17, the stretch for GDSTR-3D remains low and is between that for the sparse (average node degree 7) and dense networks (average node degree 16) with no obstacles. The stretch for both AODV and VRR seem to be unaffected by obstacles and remain low and independent of network size. Overall, the performance of GDSTR-3D, AODV and VRR are comparable.

5.2.3 Storage Costs

The maximum storage requirement per node for low density networks is shown in Figure 18. We omit the corresponding graph for high density networks because the trends are similar. From these results, we find that the storage cost of GDSTR-3D increases very slightly with increasing network size and this trend seems to hold true for all the different aggregation methods. The maximum storage required for 2×2D hulls remains below 1,000 bytes even for 3,200-node networks. The storage cost for S4 grows somewhat faster than GDSTR, but it still remains less than 2,000 bytes for 3,200-node networks. In contrast, the storage cost for VRR is significantly higher.

The asymptotic behaviors stem from how each algorithm stores the information required for routing. GDSTR only needs to store information about the neighbor and the aggregated convex hull, hence it requires almost constant storage space (though the hulls do tend to get larger as the network size increases). S4 needs to store the route to each of the beacons and the nodes inside the same cluster. Since the number of beacons grows in the order of $O(\sqrt{n})$, the storage cost grows in the order of $O(\sqrt{n})$ as well.

For VRR, each node maintains an entry for each virtual path that includes the node. While DHTs typically maintain only $O(\log n)$ routing entries per node, each VRR node maintains much more state than $O(\log n)$ since they are required to also maintain entries for the virtual paths on which it is an intermediate node. The number of virtual paths scales up significantly with increasing network size and hence the storage requirement also increases correspondingly. A contributing factor for the increase in storage requirement between 200 and 400 nodes is that once we exceed 256 nodes, we need two bytes instead of one to address each node.

5.2.4 Control Traffic Overhead

We plot the total bandwidth required per node to converge for low density networks in Figure 19. Note that the lines for GDSTR-2D and GDSTR-3D (sphere) are so close to the x-axis that they are barely visible. As before, we omit the corresponding graph for high density networks because the trends are similar.

Among the algorithms investigated, GDSTR-3D incurs the slowest rate of growth in maintenance bandwidth as the network size n scales up. GDSTR-3D incurs a bandwidth cost of approximately $O(\log n)$. The main source of the overhead is the aggregation of hulls, where each update from a child node will trigger an update message for the parent. Hence, the cost increases with the depth of the tree, i.e. the bandwidth cost has order of growth $O(\log n)$. These results also show that for large networks, GDSTR-3D (2×2D) in-

curs a significant increase in maintenance bandwidth compared to GDSTR-3D (sphere) and GDSTR-2D.

For S4, because each node needs to know its distance to every beacon, the complexity grows with the number of beacons, which has order of growth $O(\sqrt{n})$. There is a significant difference between the current implementations of S4 and GDSTR-3D. S4 attempts to aggregate messages from multiple nodes before broadcasting an update; for GDSTR-3D, a node sends out an update message immediately after computing its hull using the latest update from a child node. This implies that the aggregation algorithm for GDSTR-3D can be further optimized to reduce the bandwidth requirement. Delaying the updates will however increase the convergence time of the network, which is why we chose not to implement this feature in our current implementation.

Compared to S4 and GDSTR-3D, VRR seems to incur more message overhead. This is because the length of the virtual paths are not fixed and can be arbitrarily long. Hence in the construction and tearing down of the paths, we found it difficult to select an appropriate timeout period. If the period is too short, some of the paths might not have converged; if it is too long, the algorithm would not be sensitive to topology changes. As such, we cannot yet make a conclusive statement about its scalability, though the signs seem to suggest that it is less scalable than GDSTR-3D and S4.

6 Discussion

Comparing GDSTR-3D to VRR. We obtained the original source code for VRR from the authors of [2]. We ported the code to TinyOS v2.x and fixed a few minor bugs. From studying the source code, it seems that there might be issues with memory management and also with the current link-layer implementation. In this light, it is possible that we have not compared our algorithm to the "optimal" implementation of VRR. Our results do however suggest that VRR cannot achieve the same level of routing efficiency as S4 and GDSTR-3D. This is actually not entirely surprising since it is well-known that the virtual hops for DHTs do not map well to physical proximity. Also because the VRR routing algorithm is a form of greedy forwarding, some amount of inefficiency is to be expected since VRR does not naturally exploit proximity information of the network topology like S4 and GDSTR-3D.

It is worth noting that our goal is not to show that GDSTR-3D is better than other point-to-point routing algorithms. Instead, we merely wish to complete a comprehensive survey of the major point-to-point algorithms proposed thus far in order to better understand their tradeoffs in the 3D sensor network setting. To the best of our knowledge, this has not been done before.

Need for coordinates & coordinate resolution. Before geographic routing algorithms can be used in a sensor network, geographic (Euclidean) coordinates must be available. For a small 100- to 200-node network, it might be possible to assign coordinates manually. For large sensor networks with thousands of nodes, manual coordinate assignment is patently infeasible. A possible solution to the coordinate assignment problem is to use 3D virtual coordinates [28].

Geographic routing algorithms also require a coordinate resolution service to map node identifiers to routing coordinates. However, we note that even without a coordinate resolution service, GDSTR-3D can be used for data-centric applications that are tied only to the coordinates, provided coordinates are available (either through manual assignment, or by computation).

S4 faces a similar challenge since a location service that maps the destination node to the nearest beacon also needs to be available before S4 can be used for routing. In our deployment, the S4 nodes obtain the coordinates of the destination from a centralized location directory through a serial port. Such a service would not be available in a typical (non-testbed) deployment, and a distributed location directory similar to GHT [24] might be required.

Furthermore, S4 assumes that beacons are selected beforehand, i.e. the beacons are distributed evenly over the entire network. However, if the network is turned on incrementally, it might be necessary to re-elect the beacons and initiate the coordinates of the nodes with respect to the beacons again as the network grows. Since the flooding phase can take more than 30 minutes, there are some feasibility concerns over such an approach.

Towards a fairer comparison of point-to-point routing algorithms. In summary, we believe that merely comparing algorithms in terms of routing efficiency and costs is not sufficient. Context and use case are also important.

While VRR and AODV might not be quite as efficient as either GDSTR-3D or S4, they route on a flat namespace which means that they can be used out of the box without the need for a location service. To the best of our knowledge, there is not much known about the cost of building a location service (for S4) or a coordinate resolution service (for geographic routing algorithms) for a sensor network. To compare GDSTR-3D and S4 fairly to VRR and AODV, it would be necessary to take into account the cost of such a location service. We leave this as future work.

7 Conclusion

We have demonstrated that existing 2D geographic routing algorithms perform badly in 3D sensor networks and that there is a need to exploit 3D coordinates for geographic forwarding in such networks to achieve good performance. Furthermore, the use of 2-hop neighbor information improves greedy forwarding success rates at little cost, since the information is already contained in the periodic stayalive messages.

We present GDSTR-3D, a new geographic routing protocol for 3D networks that uses 2-hop neighbor information during greedy forwarding to reduce the likelihood of local minima, and aggregates 3D node coordinates using two 2D convex hulls. We show in both testbed experiments and TOSSIM simulations that GDSTR-3D is able to guarantee packet delivery and achieve hop stretch close to 1. GDSTR-3D also scales well to networks of up to 3,200 nodes and has a smaller memory and maintenance bandwidth requirements compared to VRR and S4 for large networks. Given these properties, we believe GDSTR-3D is an attractive choice for point-to-point routing in large three-dimensional wireless sensor networks.

8 Acknowledgements

We would like to acknowledge the support of the Singapore Ministry of Education under the grant R-252-000-311-112 for this research. We are also grateful to Hongyang Li, our shepherd Anish Arora and the anonymous SenSys reviewers who helped us improve the paper with their insightful comments.

9 References

[1] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[2] M. Caesar, M. Castro, and E. B. Nightingale. Virtual ring routing: network routing inspired by DHTs. In *Proceedings of ACM SIGCOMM 2006*, pages 351–362. ACM Press, 2006.

[3] M. Doddavenkatappa, A. L. Ananda, and M. C. Chan. INDRIYA: A wireless sensor network testbed, 2009. http://indriya.ddns.comp.nus.edu.sg.

[4] S. Durocher, D. Kirkpatrick, and L. Narayanan. On routing with guaranteed delivery in three-dimensional ad hoc wireless networks. *Wireless Networks*, 16(1):227–235, 2010.

[5] R. Flury and R. Wattenhofer. Randomized 3D geographic routing. In *Proceedings of the IEEE INFOCOMM 2008*, 2008.

[6] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proceedings of NSDI 2005*, May 2005.

[7] P. S. Geoffrey Werner-Allen and M. Welsh. Motelab: A wireless sensor network testbed. In *Proceedings of IPSN 2005*, 2005.

[8] R. Govindan. Data-centric routing and storage in sensor networks. *Wireless sensor networks*, pages 185–205, 2004.

[9] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proceedings of REALMAN '06*, pages 63–70, New York, NY, USA, 2006. ACM.

[10] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of Mobicom 2000*, pages 243–254, Boston, MA, August 2000.

[11] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *Proceedings of NSDI 2005*, May 2005.

[12] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. On the pitfalls of geographic face routing. In *Proceedings of DIAL-M-POMC 2005*, September 2005.

[13] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Lazy cross-link removal for geographic routing. In *Proceedings of SenSys 2006*, November 2006.

[14] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[15] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proceedings of MobiHoc 2003*, June 2003.

[16] S. S. Lam and C. Qian. Geographic routing with low stretch in d-dimensional spaces. Technical report, The University of Texas at Austin, January 2010.

[17] B. Leong, B. Liskov, and R. Morris. Geographic routing without planarization. In *Proceedings of NSDI 2006*, May 2006.

[18] B. Leong, S. Mitra, and B. Liskov. Path vector face routing: Geographic routing with local face information. In *Proceedings of ICNP 2005*, November 2005.

[19] C. Liu and J. Wu. Efficient geometric routing in three dimensional ad hoc networks. In *Proceedings of INFOCOM 2009*, April 2009.

[20] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *Proceedings of NSDI 2007*, April 2007.

[21] J. Newsome and D. Song. GEM: Graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of SenSys 2003*, November 2003.

[22] C. Perkins. Ad-hoc on-demand distance vector routing. In *Proceedings of IEEE MILCOM 1997*, November 1997.

[23] C. Qing and A. Tarek. A scalable logical coordinates framework for routing in wireless sensor networks. In *Proceedings of RTSS 2004*, pages 349–358, Washington, DC, USA, 2004. IEEE Computer Society.

[24] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mobile Networks and Applications (MONET), Journal of Special Issues on Mobility of Systems, Users, Data, and Computing*, 2003.

[25] 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 sensornets. In *Proceedings of WSNA 2002*, September 2002.

[26] K. Seada, A. Helmy, and R. Govindan. On the effect of localization errors on geographic face routing in sensor networks. In *Proceedings of IPSN 2004*, pages 71–80, 2004.

[27] TelosB. TelosB datasheet. web. http://www.xbow.com.

[28] J. Zhou, Y. Chen, B. Leong, and B. Feng. Practical virtual coordinates for large wireless sensor networks. In *Proceedings of ICNP 2010*, October 2010.