Practical Virtual Coordinates for Large Wireless Sensor Networks

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

Yu Chen and Ben Leong
Department of Computer Science
National University of Singapore
{chenyu, benleong}@comp.nus.edu.sg

Boqin Feng
The School of Electronic and Information Engineering
Xi'an Jiaotong University
bqfeng@mail.xjtu.edu.cn

***Abstract*— Geographic routing is a promising approach for point-to-point routing in wireless sensor networks, but it requires the availability of geographic coordinates. Location devices like GPS do not work indoors and they are often not cost-effective for ubiquitous deployment on a large scale. While it is possible to manually configure coordinates for small sensor networks, it is infeasible to do the same for large-scale networks with thousands of nodes. We present Particle Swarm Virtual Coordinates (PSVC), a distributed virtual coordinate assignment algorithm that employs Particle Swarm Optimization to compute virtual coordinates for geographic routing. PSVC converges faster, achieves a lower hop stretch, and scales well up to large networks of 3,200 nodes compared to NoGeo. Also, PSVC makes no assumptions on the network topology and can naturally be extended to three-dimensional (3D) wireless sensor networks.**

I. Introduction

Geographic routing [8, 10, 12, 13] is a promising approach for point-to-point routing in wireless sensor networks. Geographic routing can often achieve close to optimal routing stretch [13] and scales well, typically requiring the maintenance of routing state that is dependent only on local network density and not on network size [10].

Geographic routing however requires the availability of geographic (Euclidean) coordinates. While it is possible to manually configure coordinates for small sensor networks, it is infeasible to do the same for large-scale networks with thousands of nodes. Also, location devices like GPS do not work indoors and they are often not cost-effective for ubiquitous deployment on a large scale.

The natural solution to this problem is therefore to employ virtual coordinates. In fact, this paper is motivated by our attempts at implementing previously developed virtual coordinate assignment algorithms [14, 18] for TinyOS-based motes. In our work, we found that these algorithms are typically ill-suited for deployment in a practical sensor environment because of slow convergence, implementation complexity and large compiled binary size.

In this paper, we describe a new distributed virtual coordinate assignment algorithm called Particle Swarm Virtual Coordinates (PSVC), that computes coordinates using an approximation algorithm called Particle Swarm Optimization (PSO) [9]. Like GSpring [14], PSVC elects a small number of reference nodes at the boundary of the network. Initial coordinates are then assigned using PSO to minimize the error for an objective function that maps the assigned virtual coordinates for each node to their hop counts to the reference nodes. PSVC also employs a relaxation algorithm that models the virtual coordinates as a system of nodes and springs to improve the convexity of the virtual topology.

We also found that existing two-dimensional (2D) virtual coordinate algorithms perform poorly for three-dimensional (3D) networks. Unlike NoGeo [18] and GSpring [14], PSVC does not make any assumptions on the planarity of the physical network topology and can be naturally extended to 3D wireless sensor networks. PSVC also has a faster convergence rate, achieves lower hop stretch, and scales well up to large networks of 3,200 nodes compared to NoGeo. Most importantly, our algorithm is practical and has been successfully deployed on a real TinyOS (TelosB) mote testbed.

The remainder of this paper is organized as follows: in Section II, we present an overview of existing and related work. In Section III, we describe our new algorithm. We present our simulation evaluation for 2D and 3D networks in Sections IV and V respectively and our evaluation on a real sensor testbed in Section VI. Finally, we conclude in Section VII.

II. Related Work

Rao et al. were first to propose the use of virtual coordinates for geographic routing [18]. Their algorithm,

which we refer to as NoGeo, elects a root node and subsequently a set of p perimeter nodes based on their hop counts to the root node. Subsequently, beacons are broadcast from the p perimeter nodes to all the nodes in the system and each node computes its virtual coordinates by solving an optimization problem. NoGeo however has several drawbacks: (i) for sparse networks, the greedy forwarding success rate is low and consequently, the associated hop stretch is very high; (ii) the greedy forwarding success rate is low for 3D networks and networks with obstacles with 800 or more nodes; (iii) the storage cost is high, since it needs to store the hop count between each pair of p perimeter nodes, which incurs a storage cost of $O(p^2)$. Because p has order of growth $O(\sqrt{n})$, where n is the network size, the storage cost per node is approximately $O(n)$.

There have since been several proposals for computing *greedy embeddings* for geographic routing. A greedy embedding is an embedding such that, at each vertex, there always exists at least one neighbor which is closer to the destination. In other words, it is an embedding for which greedy forwarding is sufficient to guarantee packet delivery. GSpring [14] introduced a novel spring relaxation algorithm that incrementally improves the convexity of the voids in the virtual topology, thereby improving the greedy forwarding success rate. GSpring is unfortunately impractical for implementation in practical networks because it assumes the existence of a geocast mechanism and takes about 1,000 iterations to converge, which incurs significant overhead. Kleinberg proposed a method to construct a greedy embedding in 2D hyperbolic space [11]. The greedy embedding is constructed using n bits to describe the location of the nodes. The algorithm was evaluated with a simulation for a 50-node network, and it achieves an average hop stretch of 1.2. Westphal and Pei subsequently proposed another greedy embedding in polylogarithmic-dimensional Euclidean space [20], called GLoVE, that reduces the dimensionality of greedy forwarding from n to $k = \frac{\log(n)}{\epsilon^2}$, while still respecting the distance between points within an ϵ factor. GLoVE-U avoids local minima by routing in a tree structure, with a blacklist of size $O(\log(n))$. These methods will require more than a constant number of bits for addressing which 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 we cannot afford to use too many bits for addressing.

There are other routing algorithms based on non-Euclidean coordinate systems. VPCR is a routing algorithm based on virtual polar coordinates [17]. VPCR does not perform as well as existing geographic routing algorithms and incurs significant overhead under node and network dynamics. There are also routing algorithms that use the hop counts to a set of landmark nodes (beacons) as a virtual coordinate system [1, 5, 16]. The major drawback for these algorithms is that a large number of beacons (about 30 to 40) is typically needed to achieve routing performance comparable to geographic routing algorithms and that they typically either have to resort to flooding when packets end up at local minima [5] or maintain $O(\sqrt{n})$ state [16].

III. Particle Swarm Virtual Coordinates (PSVC)

In this section, we describe the Particle Swarm Virtual Coordinates (PSVC) algorithm in detail. Like previous algorithms [14, 18], PSVC can be divided into two phases: computation of initial coordinates and relaxation.

The selection of the reference nodes and the relaxation steps are similar to GSpring [14]. The main difference lies in the way that the coordinates of the reference nodes are computed. Instead of arranging the reference nodes in a circle, we attempt to model the hop counts between the reference nodes as a spatial distance in a manner similar to NoGeo [18]. While we use a minimization equation that is similar to NoGeo, our method produces better coordinates because the perimeter detection mechanism for NoGeo is not accurate and it sometimes wrongly identifies non-perimeter nodes as perimeter nodes.

In this section, we describe PSVC for two-dimensional (2D) coordinates for simplicity. However, it should be clear that PSVC can be trivially extended to three-dimensional (3D) coordinates, while the same cannot be said for both NoGeo [18] and GSpring [14], which assume that the network topology is planar.

A. Preliminaries: Particle Swarm Optimization

In this section, we provide a brief overview of the Particle Swarm Optimization (PSO) algorithm [9] that is used to compute virtual coordinates.

PSO was motivated by the social behavior observed in bird flocks and fish schools. PSO solves an optimization problem where the goal is to find a vector $\vec{x} = (x_i, x_2, \cdots, x_D)$ that maximizes (or minimizes) an objective function $F(\vec{x})$ by simulating particles moving in a D-dimensional space. Possible solutions are modelled as the positions of n particles, where particle p_i has position $\vec{x}_i$ and velocity $\vec{v}_i$.

The particles are initialized randomly with coordinates within a bounded space and their positions are updated by their velocities. The key idea is that the flight of each particle is influenced by both its own past flight history and also the flight histories of the other particles in the system. To this end, particle $\vec{p}_i$ maintains and constantly updates the best position it has seen in the past as $\vec{l}_i$ and the system also maintains and constantly updates the global best position $\vec{G}$.

The simulation is done in iterations. In each iteration, the positions of the particles are updated according to the following equations [19]:

$$\vec{v}_i = w\vec{v}_i + c_1 r_1(\vec{l}_i - \vec{x}_i) + c_2 r_2(\vec{G} - \vec{x}_i) \quad (1)$$

$$\vec{x}_i = \vec{x}_i + \vec{v}_i \quad (2)$$

$$w = w_{max} - \frac{k}{k_{max}}(w_{max} - w_{min}) \quad (3)$$

where c_1, c_2, w_{max} and w_{min} are positive constants, r_1 and r_2 are uniform random variables over the range [0,1], k_{max} is the total number of iterations and k is the number of the current iteration.

B. Election of Reference Nodes

The nodes in the system first achieve a consensus on the node r with the smallest node identifier (ID). This is done by having each node broadcast the identifier of the node with the smallest ID currently known to it, together with its hop count to the node. When the consensus is reached, each node will know its hop count to the reference node r.

Next, the system elects the node with the largest hop count to r as the first reference node p_1 in a similar manner. The remaining reference nodes are elected similarly. In particular, the reference node p_i is defined as the node with the maximum sum of the square roots of the hop counts to the earlier reference nodes $p_j, j = 1, \cdots, i-1$. We break ties by comparing the node IDs. A side effect of this consensus-based reference node election algorithm is that all the nodes in the network will know of their hop counts to all the reference nodes when the process ends.

The election of each reference node floods the network, but the cost per reference node is approximately $O(D) \approx O(\sqrt{n})$, where D is the diameter of the network and n is the network size. PSVC uses 4 reference nodes for 2D networks and 6 reference nodes for 3D networks because we found that this achieves good performance in practice.

C. Initialization of Reference Nodes

We set the coordinates for p_1 as $(0, 0)$ and the coordinates for p_2 as $(100h_{12}, 0)$, where h_{ij} is the hop count from p_i to p_j. A scaling factor of 100 is multiplied with h_{12} to ensure that the computed coordinates are sufficiently far apart to avoid floating point rounding errors during routing. The coordinates for the third reference point $p_3 = (x_3, y_3)$, is the point which is at a distance $100h_{31}$ from p_1 and at a distance $100h_{32}$ from p_2. (x_3, y_3) can be computed with triangle equalities.

We define the following error function for the position $\vec{x}_k$ of reference node p_k:

$$E = \sum_{i=1}^{k-1}(|\vec{x}_k - \vec{x}_i| - 100h_{ik})^2 \quad (4)$$

The coordinates of the remaining reference nodes are obtained by using PSO to solve for the coordinates that minimize the error E. Note that for PSVC in 3D coordinates, there exists a solution for $\vec{x}_4$, where $E = 0$.

D. Coordinates for Non-Reference Nodes

Once each of the p reference nodes determines its initial coordinates, these coordinates are broadcast to all the nodes in the network. Each non-reference node then proceeds to compute its own coordinates from its hop counts to and the coordinates of all the reference nodes.

In the PSO simulation, we define the number of particles as $POPSIZE$ and the total number of iterations as k_{max}. To prevent floating point overflow, we normalize the coordinates by dividing them by $100h_{12}$, the virtual distance between reference nodes p_1 and p_2, so that all the dimensions will be in the range $[-1, 1]$. The details of our PSO implementation is shown in Algorithm 1. After PSO converges, we scale the coordinates of the node by multiplying the factor of $100h_{12}$. In our implementation, we set the $POPSIZE = 10$, $c_1 = c_2 = 1.8$, $w_{max} = 1.2$, $w_{min} = 0.1$ and $k_{max} = 100$. The parameter settings for PSO only determines the efficiency of the convergence and not the final result, and the parameters we chose are "reasonable" settings that seem to work in practice.

E. Relaxation after Initialization

After the initial coordinates are determined, we run an iterative relaxation procedure to make the virtual topology more convex. Like GSpring [14], we model the network as a system of nodes and springs. Each link between two neighboring nodes i and j that can communicate with each other is modeled as a spring of

Algorithm 1: Compute initial coordinates for non-reference nodes with PSO

Given $\vec{p}_i, i = 1, \cdots, p$
Initialize $\vec{x}_i \in [-1, 1]$, $\vec{v}_i \in [-1, 1]$, $\vec{G} = \vec{0}$,
$\vec{l}_i = \vec{0}, i = 1, \cdots, POPSIZE$,
$Lerror_i = \infty, i = 1, \cdots, POPSIZE$
$\vec{G} = \vec{0}$, $Gerror = \infty$
for $k = 0$ to k_{max} **do**
  $w = w_{max} - \frac{k}{k_{max}}(w_{max} - w_{min})$
  for $i = 0$ to $POPSIZE$ **do**
    $\vec{v}_i \leftarrow w\vec{v}_i + c_1 r_1(\vec{l}_i - \vec{x}_i) + c_2 r_2(\vec{G} - \vec{x}_i)$
    $\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$
    $error = \sum_{j=1}^{p}(|\vec{x}_i - \vec{p}_j| - \frac{h_j}{h_{12}})^2$ // where p is the number of reference nodes, h_j is the hop count from the current node to the reference node j, $\vec{p}_j$ is the position for reference node j.
    if $error < Lerror_i$ **then**
      $\vec{l}_i \leftarrow \vec{x}_i$
      $Lerror_i = error$
    end if
    if $error < Gerror$ **then**
      $\vec{G} \leftarrow \vec{x}_i$
      $Gerror = error$
    end if
  end for
end for

rest length l_{ij}, which depends on the number of common neighbors. Assume S_{ij} is the set of common neighbors for nodes i and j, S_i and S_j are the sets of neighbors not shared by the other node respectively, the percentage of common neighbors r_{ij} is defined as follows:

$$r_{ij} = \begin{cases} 0, & \text{if } |S_{ij}| + |S_i| + |S_j| = 0 \\ \frac{|S_{ij}|}{|S_{ij}|+|S_i|+|S_j|}, & \text{otherwise} \end{cases} \quad (5)$$

Note that $0 \leq r_{ij} \leq 1$ and the rest length of the spring, l_{ij}, between two nodes i and j is then given by:

$$l_{ij} = l_{max} - r_{ij}(l_{max} - l_{min}) \quad (6)$$

where l_{min} and l_{max} are constants such that $l_{min} < l_{max}$.

According to Hooke's law, the force vector $\vec{F}_{ij}$, exerted by the spring between two nodes i and j on node i, is given by the following formula:

$$\vec{F}_{ij} = k \times (l_{ij} - |\vec{x}_i - \vec{x}_j|) \times u(\vec{x}_i - \vec{x}_j) \quad (7)$$

where k is the spring constant, l_{ij} is the spring rest length, and $\vec{x}_i$ and $\vec{x}_j$ are the coordinate vectors of nodes i and j respectively. The difference between the spring rest length and the actual spring length $(l_{ij} - |\vec{x}_i - \vec{x}_j|)$ is the displacement of the spring from rest, and $u(\vec{x}_i - \vec{x}_j)$ is the unit vector from $\vec{x}_j$ to $\vec{x}_i$, specifying the direction of the force. We compute the net force on a node by vector addition:

$$\vec{F}_i = \sum_{j \neq i} \vec{F}_{ij} \quad (8)$$

A node will periodically update its coordinates based on the virtual coordinates of its immediate neighbors using the following rule:

$$\vec{x}_i = \vec{x}_i + \frac{\min(|\vec{F}_i|, \alpha_t)}{|\vec{F}_i|}\vec{F}_i \quad (9)$$

where α_t is a damping constant that decreases exponentially after time T, according to the following equation:

$$\alpha_t = \begin{cases} \alpha_{max}, & \text{if } t < T \\ \alpha_{max} e^{-\frac{t}{T}}, & \text{otherwise} \end{cases} \quad (10)$$

Here, α_{max} and T are constants and t is the number of iterations after a node starts updating its coordinates. When $\alpha < \alpha_{min}$, we stop updating the coordinates. In our implementation, we set $k = 0.5$, $\alpha_{min} = 1$, $\alpha_{max} = 3$ and $T = 20$.

As mentioned in Section III-C, we chose a basic "virtual hop" length of 100 units in the initial coordinate assignment. After experimenting with many different values, we found that the spring constants $l_{max} = 30$ and $l_{min} = 15$ yielded the best results.

F. Node Joins after Convergence

The steps described in Sections III-B to III-E are only used for the initialization of a network. Once the relaxation of the nodes has stabilized, the nodes typically do not change their coordinates since constant changes to the virtual coordinates would complicate the addressing of the nodes during routing.

When a node joins a network, it listens to the stayalive beacons of its neighbors to obtain their coordinates. If it finds that the coordinates of its neighbors have not stabilized, it computes its coordinates according to the process described in Section III-D. Otherwise, if all its neighbors have stabilized, it computes its coordinates as a weighted sum of the coordinates of its neighbors' coordinates as follows:

$$x_i = \frac{1}{\sum_j r_{ij}} \sum_j r_{ij} x_j \quad (11)$$

$$y_i = \frac{1}{\sum_j r_{ij}} \sum_j r_{ij} y_j \quad (12)$$

where r_{ij} is the percentage of common neighbors for nodes i and j as defined in Equation (5).

IV. Simulation Evaluation for 2D networks

In this section, we evaluate the performance of PSVC by comparing it to NoGeo [18] and the actual physical coordinates with simulations in TOSSIM, a simulator for TinyOS. We investigated the effect of network density, network size and also obstacles. The error bars for all the plots in our evaluations indicate the 95% confidence intervals.

In our TinyOS implementation of PSVC, each reference node will periodically broadcast a heartbeat message. The period t (in seconds) between each broadcast is uniformly distributed in the range $[3, 3n]$, where n is the number of reference nodes. We set $n = 4$ for 2D networks and $n = 6$ for 3D networks in our implementation because we found that this strikes a balance between performance and cost. Each heartbeat message contains the ID of the reference node, hop count to the reference node and the sequence number of the message. A node will discard the messages whose sequence numbers are superseded by other messages, and broadcast the remaining messages to its neighbors. Each node will compute its initial coordinate after it learns of its hop count to all the reference nodes. Once a node has computed its initial coordinates, it will broadcast its coordinates in its own stayalive message every 2 seconds. After a node finds that all its neighbors have initialized their coordinates, it starts to update its coordinates by running the relaxation algorithm described in Equation (9). NoGeo is implemented according to its description in [18].

A. Effect of Network Density

In the simulation, we generate our topology using a simple radio model: all nodes have the same radio range and for a given radio range, two nodes can communicate if and only if they are within range. Nodes are randomly generated in a plane of size $2,000 \times 2,000$ and the network densities are varied by adjusting the radio range. To ensure that the generated topologies are connected, we repeatedly add nodes at random and remove the smaller connected components until there is only one large connected component with the desired number of nodes (200). We generated 50 random topologies for each density. In our simulations, we ignore the link-layer errors since our focus is to evaluate the efficiency and cost of our algorithm in the routing layer. The impact of the link layer is evaluated in a real sensor testbed in Section VI.

To measure the hop stretch, which is defined as the ratio between the actual routing path and the optimal routing path, we compared the performance of GDSTR [13] on different virtual coordinate systems and also on the actual physical coordinates. GDSTR first attempts to forward a packet to the destination using greedy forwarding if there exists a neighbor nearer to the destination. When greedy forwarding fails, it resorts to routing along a spanning tree in order to guarantee packet delivery. Since the greedy forwarding mode is usually much more efficient than tree forwarding [21], greedy forwarding success rate is also an important benchmark for virtual coordinate systems. GDSTR was implemented mostly as described in [13], with some minor differences because TinyOS does not support dynamic memory allocation. In our simulation experiments, each node acts as a source and sends a packet to a randomly-selected destination node every 50 ms over a period of 30 minutes (1,800 s) after the network is allowed to stabilize for 30 minutes.

1) Greedy Forwarding Success Rate: In Fig. 1, we compare the greedy forwarding success rates for different virtual coordinate algorithms with either a one-hop or two-hop neighbor forwarding strategy. The use of two-hop neighbor information for greedy forwarding incurs little additional cost, since the required information is already contained in the periodic stayalive messages.

We make three key observations: first, the greedy forwarding success rates for virtual coordinates (both PSVC and NoGeo) are higher than that for physical coordinates for sparse networks (average node degree below 8); second, as expected, two-hop greedy forwarding can improve the greedy forwarding success rate compared to one-hop greedy forwarding and the improvement is most pronounced for sparse networks, especially for physical coordinates; finally, PSVC performs better than both physical coordinates and NoGeo for sparse networks and achieves close to 100% greedy forwarding success rates for dense networks. Given that these results were obtained using only 4 reference nodes for PSVC, we conclude that 4 reference nodes are sufficient.

2) Average Hop Stretch: In Fig. 2, we plot the hop stretch for GDSTR running over various virtual coordinate systems. As predicted by the greedy forwarding success rates shown in Fig. 1, the hop stretch for dense networks is close to 1 for GDSTR over both PSVC and physical coordinates. What is interesting is that PSVC performs better than physical coordinates in sparse networks. The reason for this is that because of the relaxation process, PSVC has a tendency to generate virtual topologies that are somewhat more convex than randomly-generated sparse topologies. When the network is more convex, there are fewer local minima and because greedy forwarding success rate is higher,

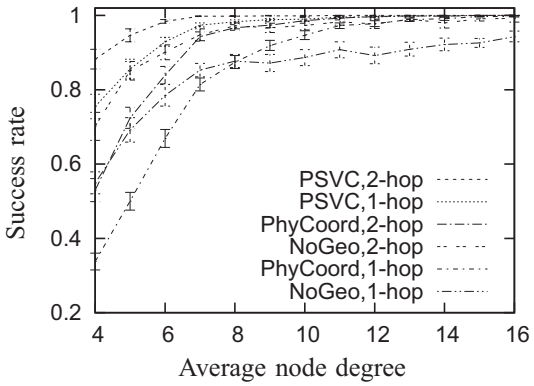


Fig. 1. Plot of greedy forwarding success rate against network density.

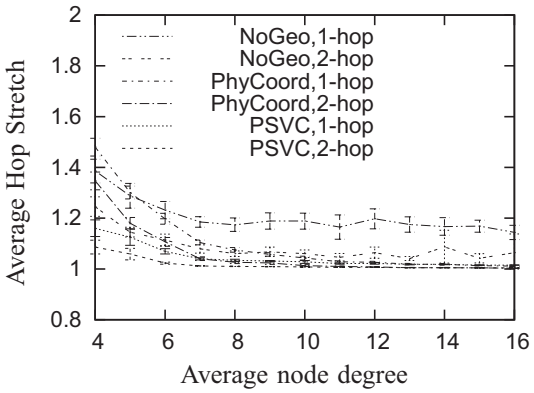


Fig. 2. Plot of average hop stretch for GDSTR against network density.

the overall geographic routing performance is better.

We found that NoGeo does not perform as well as PSVC for sparse networks and it performs especially poorly for dense networks. We found that it was because NoGeo sometimes does not initialize the coordinates of nodes properly. NoGeo's perimeter node detection algorithm tends to cause perimeter nodes to cluster on one side of the perimeter in the physical topology and this has implications on the computations for the initial coordinates. The NoGeo relaxation process helps somewhat, but only marginally. This is clear from the virtual coordinates derived for the donut-like and cross-shaped networks shown in Figs. 3 and 4 respectively. In some cases, NoGeo's perimeter detection algorithm might also wrongly classify some non-perimeter nodes as perimeter nodes.

B. Scaling Up

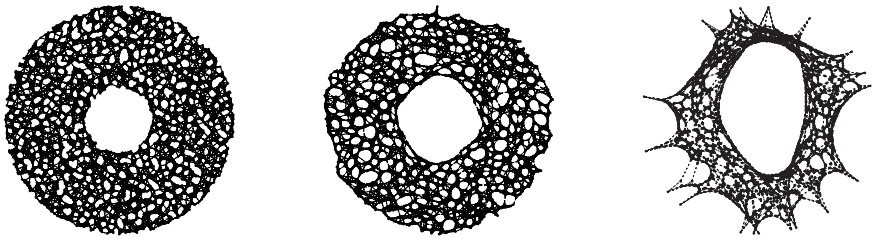
The key motivation of our work is to develop a distributed virtual coordinate assignment algorithm for large networks since it is infeasible to manually assign coordinates for large networks. It is therefore important to evaluate the performance and cost of PSVC when the network size scales up.

To this end, we generated a range of random network topologies between 50 to 3,200 nodes. The topologies with 200 or fewer nodes were generated in a $2,000 \times 2,000$ square region, while bigger networks were generated in a $3,000 \times 3,000$ square region. A total of 50 random topologies were generated for each network size.

We generated three classes of random networks: (i) sparse networks (average node degree 10), (ii) dense networks (average node degree 16) and (iii) networks with obstacles (average node degree 8). The radio range was varied in order to achieve the desired density. The networks with obstacles were generated by adding $\sqrt{\frac{n}{50}}$ cross-shaped obstacles to n-node topologies. Each cross consists of two perpendicular lines intersecting in the middle. The width of the crosses were $\frac{1}{4}$ of the region width for all network sizes.

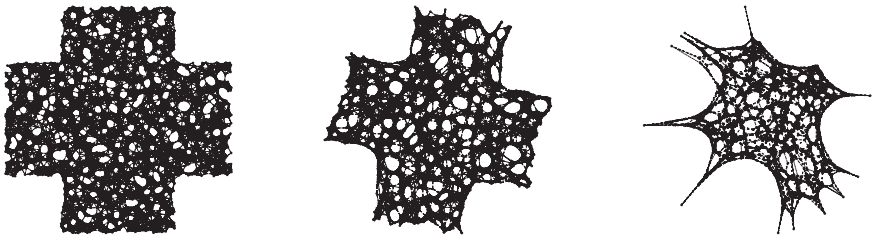
1) Greedy Forwarding Success Rate: Since a node can obtain two-hop neighbor information from the stay-alive messages of its neighbors, in this and subsequent sections, the results for greedy forwarding success rates reflect the success rates using two-hop neighbor information for various coordinate systems. We omit the results for one-hop greedy forwarding because of space constraints, and because they are similar and do not yield additional insight. In Fig. 5, we plot the two-hop greedy success rates for both dense and sparse 2D networks. We observe that for dense networks, both physical coordinates and PSVC can achieve 100% forwarding success. For sparse networks, PSVC also achieves almost 100% forwarding success, with the success rate falling slightly with increasing network size. Even at 3,200 nodes, the success rate stays above 99%. NoGeo performs worse than both PSVC and physical coordinates in all cases, though forwarding success rates remain above 90% in the range of topologies evaluated.

2) Average Hop Stretch: In Fig. 6, we plot the average hop stretch achieved by GDSTR for different virtual coordinate systems. These results are not surprising since hop stretch is directly correlated with greedy forwarding success rates. Routing performance with NoGeo is worse than that with both PSVC and physical coordinates in both dense and sparse networks. GDSTR with NoGeo achieves an average hop stretch between 1.05 and



(a) Actual (b) PSVC (c) NoGeo

Fig. 3. Actual and virtual topologies for a sample 3,200-node donut network.



(a) Actual (b) PSVC (c) NoGeo

Fig. 4. Actual and virtual topologies for a sample 3,200-node cross network.

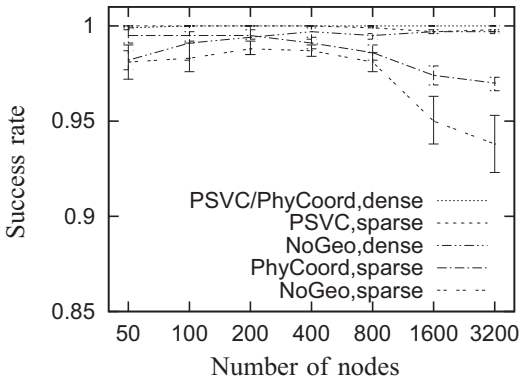


Fig. 5. Plot of two-hop greedy forwarding success rate against network size in 2D networks (average node degree 16 for dense networks, average node degree 10 for sparse networks).

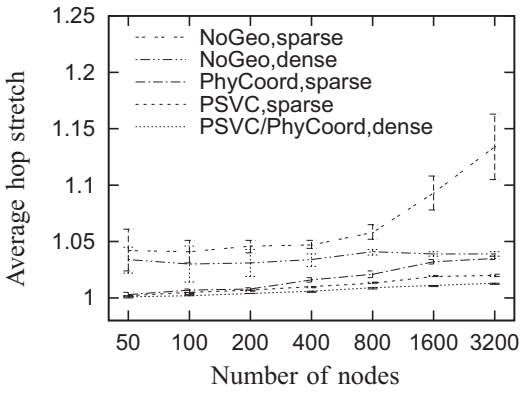
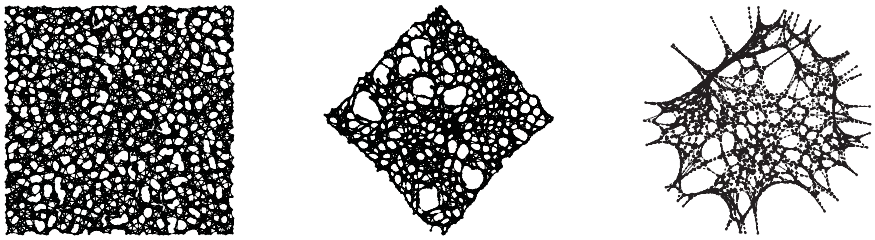


Fig. 6. Plot of average hop stretch against network size in 2D networks (average node degree 16 for dense networks; average node degree 10 for sparse networks) with GDSTR .

1.15; GDSTR with either PSVC or physical coordinates achieves a hop stretch close to 1.

3) Networks with Obstacles: In Figs. 7 and 8, we plot the routing performance for networks with obstacles. The addition of obstacles has a significant impact on greedy forwarding success rates for both physical coordinates and NoGeo. This is because the addition of obstacles significantly increases the frequency that local minima are encountered during greedy forwarding. The two-hop greedy forwarding success rates for physical coordinates and NoGeo drop to around 40% when there are 3,200 nodes; PSVC is somewhat more resilient and drops only to 80%. These results suggest that PSVC will likely work better than NoGeo in practice since practical network topologies are likely to have local minima arising from obstacles blocking some line-of-sight transmissions.

In Fig. 9, we plot the resulting network topologies for PSVC and NoGeo for a 3,200-node network with obstacles. It is clear that the topology produced by PSVC is more "similar" in shape to the actual physical network topology, and also more convex. The convexity arises



(a) Actual (b) PSVC (c) NoGeo

Fig. 9. Actual and virtual topologies for a sample 3,200-node network with obstacles.

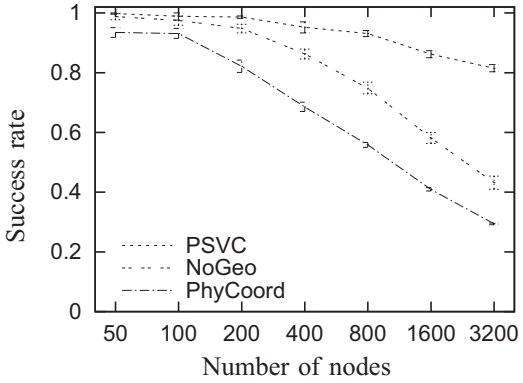


Fig. 7. Plot of two-hop greedy forwarding success rate against network size for 2D networks with obstacles (average node degree 8).

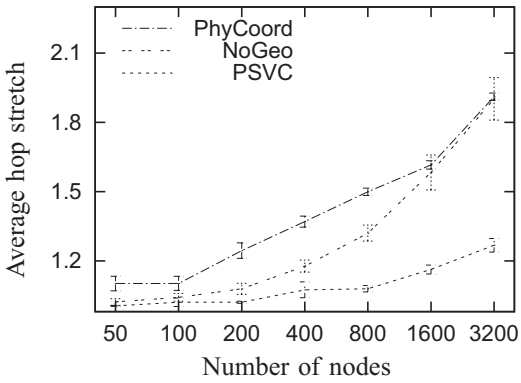


Fig. 8. Plot of average hop stretch for GDSTR against network size for 2D networks with obstacles (average node degree 8).

from the spring relaxation algorithm employed by PSVC.

4) Convergence Rate: Nodes perform relaxation for both NoGeo and PSVC. In Fig. 10, we plot the one-hop greedy forwarding success rates for the two algorithms for both dense and sparse networks with 3,200 nodes. The corresponding two-hop greedy forwarding success rates in both cases are higher. We plot the lower value because it illustrates the difference in the convergence rates between PSVC and NoGeo more clearly. It is quite clear that PSVC converges more rapidly.

For dense networks, PSVC converges rapidly to 100% greedy forwarding success rate; for sparse networks, PSVC also converges rapidly (in 50 iterations), but only to an asymptotic value of about 98%. By using PSO to compute the initial coordinates, NoGeo also converges rapidly. However, its greedy forwarding success rate is lower than PSVC. An interesting observation is that the greedy forwarding success rate for NoGeo drops after a certain threshold, so more iterations does not necessarily yield a better virtual topology for NoGeo. The same behavior can also be seen in Figure 10 of [18].

5) Storage Cost: In Fig. 11, we plot the maximum storage requirement for the networks corresponding to the results in Fig. 6 and Fig. 8. These figures include the storage required by GDSTR. We see that the storage requirements for both NoGeo and PSVC are relatively low and scale up slowly with network size, so storage is not a major concern.

6) Overhead: Similarly, in Figs. 12 and 13, we plot the message overhead for the networks corresponding to the results in Fig. 6 and Fig. 8. The message overhead for NoGeo includes stayalive messages, bootstrap node broadcast messages, perimeter node detection messages, perimeter vector broadcast messages and GDSTR setup messages. The message overhead for PSVC includes stayalive messages, reference node election messages, reference node broadcast messages and GDSTR setup messages. We make two observations: (i) the overhead for both PSVC and NoGeo seems somewhat independent of density and (ii) PSVC requires about twice as many messages to converge as NoGeo for small networks, but NoGeo rapidly catches up PSVC for large networks. The reason is that as the network size increases, the number of perimeter nodes for NoGeo increases, whereas

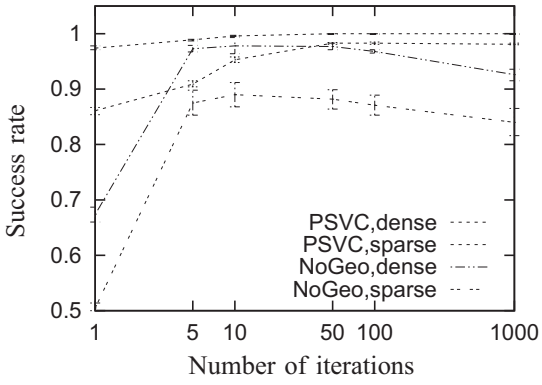


Fig. 10. Plot of one-hop greedy forwarding success rates against number of iterations for 3,200-node 2D networks (average node degree 16 for dense networks, average node degree 10 for sparse networks).

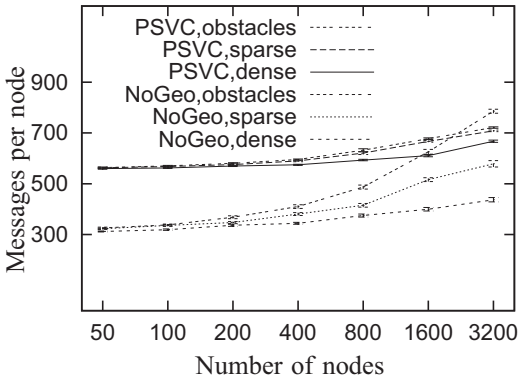


Fig. 12. Plot of messages sent per node against network size for 2D networks (average node degree 16 for dense networks, average node degree 10 for sparse networks) with GDSTR.

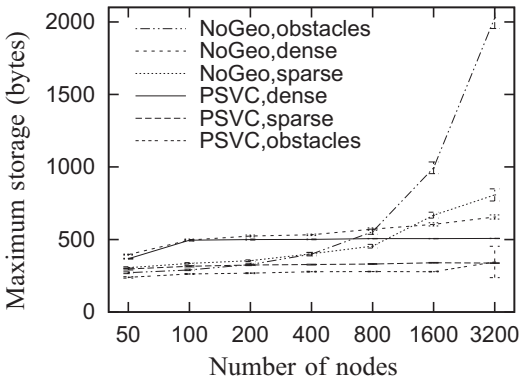


Fig. 11. Plot of maximum storage cost versus network size for various algorithms for 2D networks (average node degree 16 for dense networks, average node degree 10 for sparse networks).

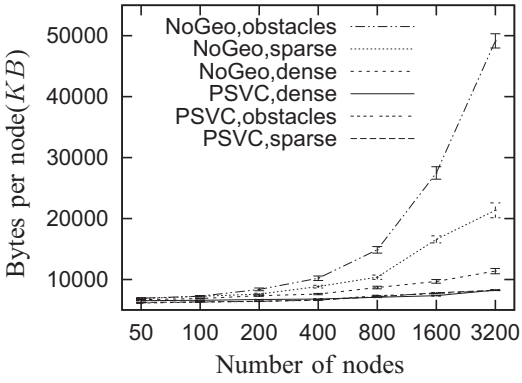


Fig. 13. Plot of bytes sent per node against network size for 2D networks (average node degree 16 for dense networks, average node degree 10 for sparse networks) with GDSTR.

PSVC uses a constant number of reference nodes. We believe that the implementation of the PSVC reference node election algorithm can be optimized to reduce the message overhead.

V. Simulation Evaluation for 3D Networks

In recent times, we have seen a number of practical deployments of three-dimensional (3D) sensor networks [2, 6, 7], and also a corresponding interest in geographic routing algorithms for 3D networks [3,4,15]. In this light, we decided that it would be helpful to evaluate the performance of PSVC and NoGeo for 3D networks.

We used experimental settings similar to those for 2D networks. We generated topologies in a $2,000 \times 2,000 \times 2,000$ cubic region for networks with 200 nodes or fewer, and a $3,000 \times 3,000 \times 3,000$ cubic region for larger networks. We investigated sparse networks with average node degree 8 and dense networks with average node degree 16. We generated 50 random networks for each density with sizes ranging from 50 to 3,200 nodes.

In Fig. 14, we plot the two-hop greedy forwarding success rates for various algorithms in both sparse and dense 3D networks. PSVC-2D uses only xy-coordinates, while PSVC-3D uses xyz-coordinates. Our first observation is that 2D virtual coordinate systems perform poorly in a 3D network. In particular, PSVC-2D achieves a two-hop greedy forwarding success rate of less than 60% in the worst case, for both sparse and dense networks. NoGeo performs even worse. The low greedy forwarding success rates render them impractical for geographic routing in 3D networks.

The second observation is that PSVC-3D performs

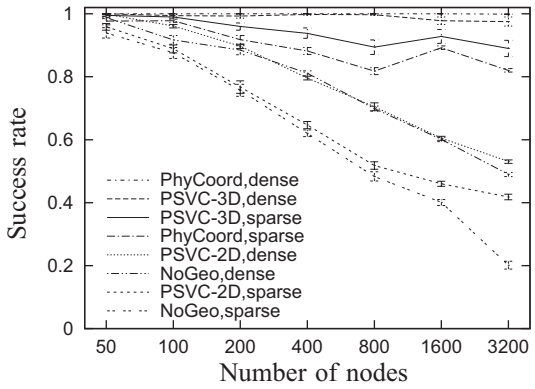


Fig. 14. Plot of two-hop greedy success rate against network size for 3D networks.

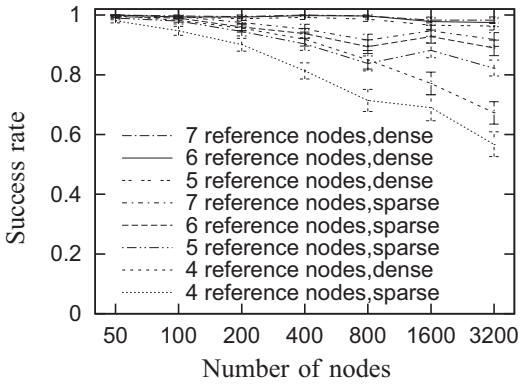


Fig. 15. Comparison of greedy forwarding success rates for PSVC with 4, 5, 6 and 7 reference nodes for 3D networks.

much better than physical coordinates in sparse networks, and has a similar performance to physical coordinates in dense networks. Its greedy forwarding success rate is greater than 95% in the worst case. As evident from the results in Section IV, there is an inverse relationship between hop stretch and greedy forwarding success rate. Due to space constraints, we omit the results for GDSTR hop stretch.

In Fig. 15, we compare the efficiency of PSVC in 3D networks with different numbers of reference nodes. While 4 references nodes are sufficient to achieve good performance for 2D networks, it is clear from these results that 4 reference nodes are insufficent for 3D networks, and that with 6 reference nodes, PSVC can achieve a greedy forwarding success rate of 90% even for sparse networks with 3,200 nodes.

VI. Evaluation on Sensor Network Testbed

In order to evaluate the performance of PSVC in a practical setting, we evaluated PSVC on Indriya [2], a wireless sensor network testbed deployed at the National University of Singapore. On Indriya, we had access to 120 TelosB sensor motes deployed on three levels of a building. Each mote has a TI MSP430 processor running at 8 MHz, with 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.

To evaluate the performance of different virtual coordinate 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. In each experiment, each node acts as a source and sends a packet to a randomly chosen destination every second for 30 minutes.

While we would have evaluated the efficiency of GDSTR with PSVC and NoGeo on Indriya, it turns out that the TelosB motes have only 48 KB of executable memory and the sizes of the binary executable for GDSTR/PSVC and GDSTR/NoGeo are too large to fit. In fact, NoGeo by itself does not fit either and the PSVC TelosB binary is already 44 KB in size.

In Fig. 16, we plot the two-hop greedy forwarding success rates for different algorithms. We can see that because Indriya is a 3D network, PSVC-2D does not work well when the nodes are distributed over two or more levels. It is no surprise that when the physical topology is a 3D network, we are able to generate a topology that is more amenable to greedy forwarding with a 3D virtual coordinate assignment algorithm. Surprisingly, while the PSVC topologies are more convex that the actual physical topology, two-hop greedy forwarding using the actual physical coordinates achieves a slightly higher success rate.

VII. Conclusion

In this paper, we present Particle Swarm Virtual Coordinates (PSVC), a new distributed virtual coordinate assignment algorithm that employs Particle Swarm Optimization to compute virtual coordinates for geographic routing. PSVC converges faster and achieves a lower hop stretch compared to NoGeo, and scales well up to large networks of 3,200 nodes. Also, PSVC makes no assumptions on the network topology and can naturally

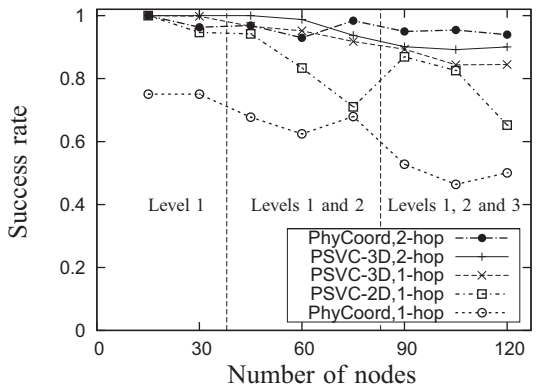


Fig. 16. Plot of two-hop greedy forwarding success rates for various algorithms on Indriya.

be extended to three-dimensional (3D) wireless sensor networks.

We have not been able to deploy PSVC and GDSTR simultaneously in the TelosB motes because the compiled binaries are too large to fit into the executable memory. We are working on improving the memory footprint of PSVC (and perhaps GDSTR if necessary) in order to fit both algorithms into the 48 KB executable memory. That said, our evaluation results have demonstrated that PSVC is a concrete step towards making geographic point-to-point routing practical for large wireless sensor networks.

ACKNOWLEDGMENT

This work was supported by the Singapore Ministry of Education grant R-252-000-311-112.

REFERENCES

[1] A. Caruso, S. Chessa, S. De, and A. Urpi. GPS free coordinate assignment and routing in wireless sensor networks. In *Proceedings of IEEE INFOCOM 2005*, March 2005.

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

[3] 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.

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

[5] 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.

[6] M. W. Geoffrey Werner-Allen, Patrick Swieskowski. Motelab: a wireless sensor network testbed. In *Proceedings in Information Processing in Sensor Networks*, 2005.

[7] 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.

[8] 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.

[9] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

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

[11] R. Kleinberg. Geographic routing using hyperbolic space. In *Proceedings of IEEE INFOCOM*, 2007.

[12] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proceedings of PODC 2003*, July 2003.

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

[14] B. Leong, B. Liskov, and R. Morris. Greedy virtual coordinates for geographic routing. In *Proceedings of ICNP 2007*, October 2007.

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

[16] 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*, 2007.

[17] 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.

[18] A. Rao, C. H. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of Mobicom 2003*, pages 96–108, San Diego, CA, September 2003.

[19] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of IEEE Conference on Evolutionary Computation*, pages 69–73, 1998.

[20] C. Westphal and G. Pei. Scalable routing via greedy embedding. In *Proceedings of IEEE INFOCOM Mini-Conference*, 2009.

[21] G. Xing, C. Lu, R. Pless, and Q. Huang. On greedy geographic routing algorithms in sensing-covered networks. In *Proceedings of MobiHoc '04*, pages 31–42, 2004.