Greedy Distance Vector Routing

Chen Qian and Simon S. Lam
Department of Computer Science, The University of Texas at Austin
Email: {cqian, lam}@cs.utexas.edu

TR-10-29 August 9, 2010

***Abstract*—Greedy Distance Vector (GDV) is the first geographic routing protocol designed to optimize end-to-end path costs using any additive routing metric, such as: hop count, latency, ETX, ETT, etc. GDV requires no node location information. Instead, GDV uses estimated routing costs to destinations which are locally computed from node positions in a virtual space. GDV makes use of VPoD, a new virtual positioning protocol for wireless networks. Prior virtual positioning systems (e.g., Vivaldi and GNP) were designed for Internet hosts and require that each host measures latencies (routing costs) to distant hosts or landmarks. VPoD does not have this requirement and uses only routing costs between directly connected nodes. Experimental results show that the routing performance of GDV is better than prior geographic routing protocols when hop count is used as metric and much better when ETX is used as metric. As a geographic protocol, the storage cost of GDV per node remains low as network size increases. GDV provides guaranteed delivery for nodes placed in 2D, 3D, and higher dimensions. We also show that GDV and VPoD are highly resilient to dynamic topology changes.**

I. Introduction

Distance Vector (DV) is a well-known routing technique. In DV forwarding, a node u chooses a physical (directly connected) neighbor v as the next hop to destination t such that it minimizes the distance $c(u,x)+D(x,t)$ for $x \in P_u$, where $c(u,x)$ is the cost of link u-x, $D(x,t)$ is the least cost from x to t, and P_u is the set of u's physical neighbors. Any *additive* metric can be used for $c(u,x)$ and $D(x,t)$. If nodes have accurate distance vectors, DV routing provides the least cost paths.

Geographic routing protocols [2] [9] [10] [15] [11] have been proposed for large-scale wireless networks because the routing state needed per node is independent of network size. Geographic routing uses greedy forwarding as the basis, i.e., a node u selects, as the next hop to destination t, a physical neighbor v that minimizes $d(x,t)$ for $x \in P_u$, where $d(x,t)$ is the physical distance between x and t. Traditionally, geographic routing protocols were evaluated using hop count or physical distance as the routing metric. De Couto *et al.* [4] showed that hop count provides poor throughput for routing in multi-hop wireless networks. Instead, other routing metrics have been proposed (e.g., ETX [4] and ETT [6]) to incorporate the effects of link loss, capacity, asymmetry, and interference.

Normalized ADVance (NADV) [14] was proposed to make geographic routing aware of link cost. In NADV, a node u selects, as the next hop to destination t, a physical neighbor v that maximizes $\frac{d(u,t)-d(x,t)}{c(u,x)}$ for $x \in P_u$, which captures the trade-off between the link cost and decrease in physical distance. A similar idea, PRR×distance, was proposed [20] for energy efficiency. Both PRR×distance and NADV improve geographic routing by making it aware of link cost. However, they only consider the cost of the next hop, and do not have routing cost information for the remaining path. Another remark about existing geographic routing protocols is that localization protocols are needed to obtain node location information. Localization protocols incur extra costs and may have large location errors.

In this paper, we present Greedy Distance Vector (GDV), the first geographic routing protocol designed with the objective of providing near-optimal paths for *any additive routing metric*. To apply GDV, each node assigns itself a virtual position (position in a virtual space) such that the Euclidean distance between any pair of nodes in the virtual space is a good estimate the routing cost between them. Like DV routing, GDV selects as the next hop to destination t, a neighbor v that minimizes $c(u,x)+\tilde{D}(x,t)$ for $x \in P_u$, where $\tilde{D}(x,t)$ is the estimated routing cost from x to t from *locally computing* the distance between the virtual positions of x and t.[1] Since $c(u,v)+\tilde{D}(v,t) \approx c(u,v)+D(v,t)$, the quality of GDV paths is expected to be close to that of optimal DV paths.

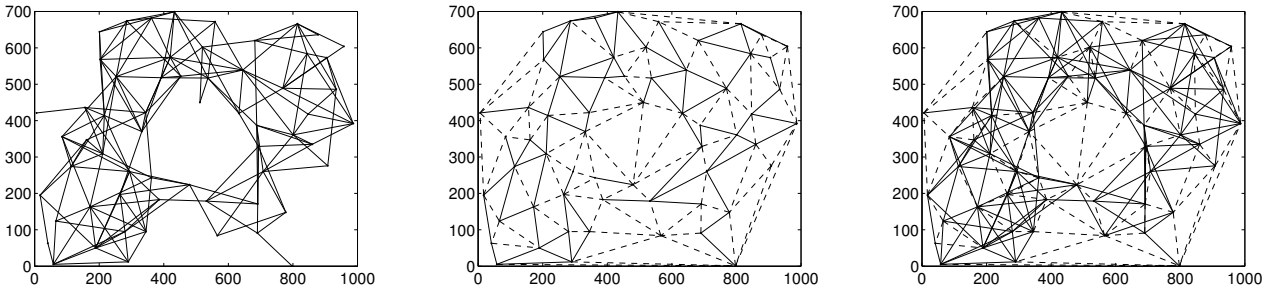
Many virtual coordinate schemes have been proposed for wireless networks when node location information is unavailable [19] [7] [3] [17] [22] [16] [21]. Unlike GDV, these schemes were not designed to predict routing costs between nodes. Instead their main objective is to improve greedy delivery rate.

The idea of embedding latencies (routing costs) in a virtual space was used by many virtual positioning systems, such as, GNP [18] and Vivaldi [5]. These systems, however, were designed for hosts with Internet routing support. More specifically, GNP requires that each node makes RTT measurements to a set of *landmark* nodes (some may be far away). Vivaldi requires that each node receives latency measurements from distant nodes from time to time.[2] In wireless networks, each node can communicate directly with its physical neighbors. But to communicate with distant nodes, GNP and Vivaldi require any-to-any routing support. Therefore, they cannot be

To appear in *Proceedings of 31st IEEE ICDCS*, Minneapolis, Minnesota, June 2011 (this version was revised, March 2, 2011)

[1] For GDV, the neighbor set of node u will be generalized to include both physical and Delaunay triangulation neighbors.

[2] In this paper, a node is *distant* iff the routing cost to that node is high.



(a) Connectivity graph (b) DT graph (c) MDT graph

Fig. 3. An illustration of connectivity, DT, and MDT graphs of a set of nodes in 2D

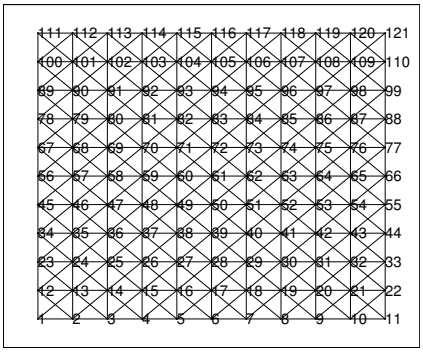
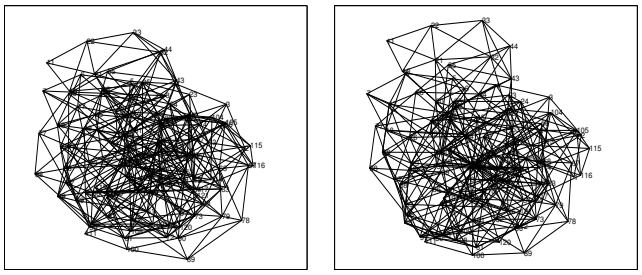


Fig. 1. 121-node network in 2D physical space



(a) After 10 adjustment periods (b) After 20 adjustment periods

Fig. 2. Virtual positions constructed by 2-hop Vivaldi

used to support GDV in wireless networks.

To illustrate the point that Vivaldi requires routing cost measurements to distant nodes, consider the 121-node grid network shown in Figure 1. Each node is only aware of its local connectivity and has no location information. We enhance the Vivaldi algorithm [5] with routing support such that it can *sample* (measure routing cost to) two-hop neighbors as well as physical neighbors. In each *adjustment period*, a node samples random nodes from its set of one-hop neighbors 100 times and its set of two-hop neighbors 100 times. Figure 2 shows the virtual positions of the nodes after 10 and 20 adjustment periods (hop count was used as routing metric). We found that almost every node is close to its physical neighbors in the virtual space. However, two nodes that are separated by many hops may also be very close in the virtual space (such as, many of the nodes near the center). Generally, there are two kinds of relationships needed for virtual positions to predict routing costs accurately [5]:

- *Local relationships*: nodes with low cost should be nearby in the virtual space.
- *Global relationships*: nodes with high cost should be far away in the virtual space.

Clearly, in this example, two-hop Vivaldi performs well for local relationships but poorly for global relationships. Routing support to sample two-hop neighbors is not sufficient for Vivaldi to work properly for wireless networks.

In this paper, we present a novel virtual positioning protocol for wireless networks, named *Virtual Position by Delaunay* (VPoD). VPoD makes use of the *multi-hop Delaunay triangulation* (MDT) of a set of nodes located in a Euclidean space, which provides a *distributed* MDT graph for routing [11]. To illustrate a MDT graph, consider Figure 3(a), which shows a connected graph of nodes and physical links (to be referred to as a *connectivity graph*). Figure 3(b) shows the Delaunay triangulation (DT) graph computed from the locations of nodes in Figure 3(a) [8]. In the DT graph, the dashed lines denote DT edges between nodes that are not connected by physical links. The MDT graph of the connectivity graph in Figure 3(a) is illustrated in Figure 3(c). By definition, the MDT graph includes every physical link in the connectivity graph and every edge in the DT graph.

For nodes in 2D, Bose and Morin proved that greedy routing in a DT always finds a given destination node [1]. Lee and Lam [12] generalized their result and proved that in a d-dimensional Euclidean space ($d \geq 2$), given a destination location ℓ, greedy routing in a DT always finds a node that is closest to ℓ. This *guaranteed delivery* property holds for node locations specified by arbitrary coordinates.

MDT protocols [11] were designed for a set of nodes to construct and maintain a multi-hop DT such that each node, say u, knows all of its DT neighbors and, if a DT neighbor, say v, is not a physical neighbor of u, a forwarding path (virtual link) is set up between u and v for them to exchange messages. Each packet is forwarded greedily at a node u as follows (to be referred to as ***MDT-greedy***): For a packet with destination d, if u is not a local minimum, the packet is forwarded to a physical neighbor of u closest to d; else, the packet is forwarded, via a virtual link, to a multi-hop DT neighbor closest to d.

For a set of nodes that maintain a correct multi-hop DT, given a destination location ℓ, it is proved that MDT-greedy always succeeds to find a node that is closest to ℓ, for nodes located in 2D, 3D, and higher dimensions with node locations specified by arbitrary coordinates [11]. In this paper, we leverage this property to design VPoD, a virtual positioning protocol for wireless nodes. Given a set of nodes running VPoD, the nodes are initially located in a virtual space (2D, 3D, or higher dimension) fairly arbitrarily. Each node u then uses an adjustment algorithm to move its position in the virtual space by comparing $D(u,v)$, its routing cost to node v in the multi-hop DT, with $\tilde{D}(u,v)$, the distance between u and v in the virtual space, where node v is either a physical neighbor or a multi-hop DT neighbor of u. Any additive metric can be used for routing costs. While prior virtual positioning protocols require routing cost information from some distant nodes to be effective, we discovered that VPoD performs very well using just routing costs from a node to its physical and DT neighbors; such cost information can be piggybacked in MDT protocol messages exchanged by neighbors.

The contributions of this paper are the following:

- GDV is the *first* geographic routing protocol designed to optimize end-to-end path costs using any *additive routing metric*, in particular, routing metrics that capture network and link characteristics other than physical distances.
- GDV and VPoD are designed for wireless networks without location information. Therefore, no localization protocol is needed.
- As a geographic protocol, GDV's storage cost per node remains low as network size (N) increases. Routing cost estimates are computed locally using virtual positions. Unlike DV, there is no need for nodes to exchange distance-vector messages of size $O(N)$.
- VPoD performs very well in preserving both local and global distance relationships between nodes in the virtual space. Every node runs the same protocols. VPoD does not require special nodes, such as, beacons and landmarks, and does not use flooding.

The balance of this paper is organized as follows. In Section II, we present the VPoD protocol. In Section III, we present the GDV protocol. In Section IV, we present experimental results to evaluate the performance of VPoD and GDV with respect to the use of adaptive timeouts, virtual space dimensionality, presence of large obstacles in physical space, and scalability to a large number of nodes. We compare the routing performance of GDV versus some existing geographic routing protocols. We also compare the quality of virtual positions provided by VPoD versus 2-hop Vivaldi. Lastly, we show that GDV and VPoD are highly resilient to dynamic topology changes. We conclude in Section V.

II. Virtual Position Construction

We next present the Virtual Position by Delaunay (VPoD) protocol to construct virtual positions for routing cost prediction in a wireless network. Each node only knows the link costs to its physical neighbors. The routing metric can be any one that is additive, such as, hop count, latency, ETX, and ETT, etc. Distances in the virtual space and routing costs are measured in the same units. Hence comparison, addition, and subtraction can be operated directly on distances and routing costs.[3]

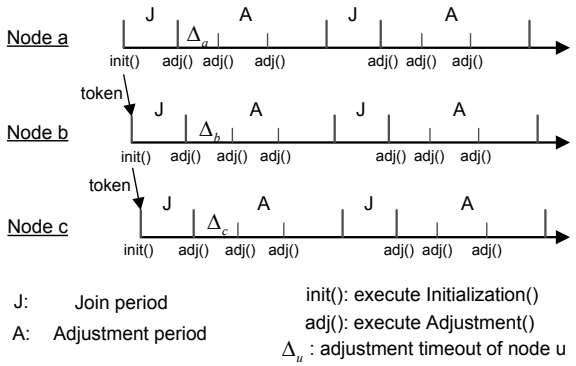


Fig. 4. Main structure of VPoD

A. Main ideas of VPoD

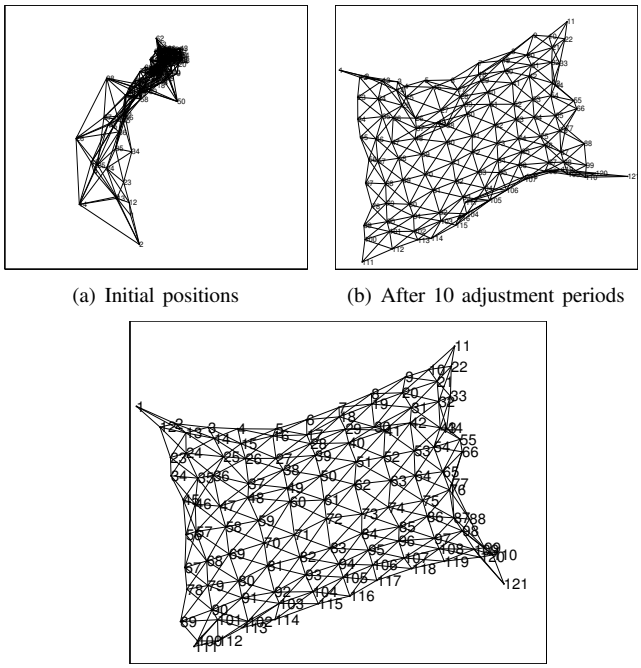
To start VPoD, an arbitrary starting node generates a token and broadcasts the token to the network.[4] Each node with a token runs the VPoD protocol. It first initializes its position in the virtual space by a simple algorithm. It then runs MDT protocols to participate in the construction of a multi-hop DT of the nodes using their locations in the virtual space. We modified the MDT protocols to record routing costs from each node to its multi-hop DT neighbors. Each node then iteratively adjusts its position by checking the positions of its physical and multi-hop DT neighbors to reduce prediction errors. For a node u, VPoD provides two types of adjustments:

1) Adjustments with physical neighbors to preserve local relationships: If its distance to a physical neighbor v is larger than its link cost to v, u adjusts its position so that its distance to v is smaller.
2) Adjustments with DT neighbors to preserve global relationships: If its distance to a multi-hop DT neighbor v is smaller (larger) than the routing cost from u to v, u adjusts its position so that its distance to v is larger (smaller).

The main structure of VPoD is presented in Figure 4. Each node runs the protocol upon receiving a token. After initializing its position, it runs the MDT join and maintenance protocols during a period of time, called *J* period, to participate in constructing the multi-hop DT. In the subsequent adjustment period, called *A* period, the node executes the adjustment algorithm iteratively to change its position in the virtual space. The multi-hop DT needs to be re-constructed after several adjustment iterations because many nodes may have changed their positions. In this manner, each node alternates between execution of the MDT protocols and the adjustment algorithm.

[3]Hereafter, whenever we say *distance*, we refer to the Euclidean distance between two nodes in the virtual space rather than the physical distance between the nodes.

[4]The starting node may be selected by any leader election protocol using a simple criterion, e.g., largest ID. Duplicate tokens received by nodes are ignored and do not affect VPoD.



(a) Initial positions (b) After 10 adjustment periods

(c) After 20 adjustment periods

Fig. 5. Virtual positions constructed by VPoD

The initialization, MDT join and maintenance protocols, and adjustment algorithm will be described in more detail in the sections to follow.

Note that after receiving the initial token, each node runs asynchronously. Different nodes may start their J and A periods at slightly different times. Each node uses its own timer to control the beginning and end of each period. After an adjustment execution, each node sends its new virtual position and estimated error to its physical neighbors and multi-hop DT neighbors. After a number of alternating J and A periods, the node positions in the virtual space converge and distances can be used to predict routing costs between nodes. The MDT protocols are then run one more time to update the multi-hop DT. VPoD does not require any landmark or perimeter node and uses no flooding. Every node in the network runs the same VPoD protocol.

We ran VPoD for the 121-node grid network in Figure 1. The results are shown in Figure 5. Note that the initial node positions are quite arbitrary. After 10 adjustment periods, the topology in the virtual space looks similar to that in the physical space. After 20 adjustment periods, all local and global relationships are preserved; compare Figure 5(c) with Figure 1 where nodes are numbered. Note that adjustment periods for VPoD and 2-hop Vivaldi are defined differently. Experimental results in Figure 14 (to be presented) show that 2-hop Vivaldi uses much more storage and communication costs per adjustment period than VPoD.

B. Position initialization

After receiving a token, each node initializes its position in the virtual space before forwarding the token to others. In our current implementation, the initial position of node u is determined as follows.

- If u is the starting node, u sets its position to the origin. Otherwise, at least one physical neighbor of u has initialized its position, namely, the token's sender.
- If only one physical neighbor, say v, of u has initialized its position, u sets its position at a random position on the circle or sphere centered at v. The radius is the link cost between u and v.
- If two or more physical neighbors of u have initialized their positions, u chooses the two that are farthest apart, and computes the mid-point between the two nodes. In order to avoid degenerate cases (three or more nodes on a line), the actual position of u is set to a random position a short distance to the mid-point.

TABLE I
NOTATION

P_u	physical neighbor set of node u
N_u	DT neighbor set of node u
F_u	forwarding table of node u
x_u	virtual position of node u, a vector
e_u	estimated position error of node u
$\hat{D}(v,t)$	Euclidean distance between the virtual positions of v and t
$c(u,v)$	cost of the link from u to v
$D(v,t)$	routing cost from node v to node t
Δ_u	adjustment timeout value of node u
c_c, c_e	tuning parameters to control the amounts of change in node position and position error

C. Multi-hop DT with extensions to support VPoD

We first briefly introduce Delaunay triangulation (DT). A *triangulation* of a set S of nodes (points) in 2D is a subdivision of the convex hull of nodes in S into non-overlapping triangles such that the vertices of each triangle are nodes in S. A DT in 2D is a triangulation such that the circumcircle of each triangle does not contain any other node inside [8]. The definition of DT can be generalized to a higher dimensional Euclidean space using simplexes and circum-hyperspheres. In each case, the DT of S is a graph to be denoted by $DT(S)$.

The model of a *distributed DT* in [12], [13] assumes that each node can directly communicate with every other node in the system. For wireless routing, the multi-hop DT model [11] was formulated as an extension of the distributed DT model as follows:

Definition 1: A multi-hop DT is specified by $\{< u, N_u, F_u > | u \in S\}$, where F_u is a soft-state forwarding table and N_u is u's set of DT neighbors locally computed by u from information in F_u.

Definition 2: A multi-hop DT of S, $\{< u, N_u, F_u > | u \in S\}$, is *correct* if and only if the following conditions hold: i) for every node $u \in S$, N_u is the same as the neighbor set of u in $DT(S)$; ii) for every neighbor pair (u,v) in $DT(S)$, there exists a unique k-hop path between u and v in the forwarding tables of nodes in S, where k is finite.

A DT neighbor of node u may be a physical neighbor if it is directly connected to u. If a DT neighbor of u is not a

physical neighbor, it is said to be a multi-hop DT neighbor. In MDT protocols, each entry in u's forwarding table F_u is a 4-tuple, $< source, pred, succ, dest >$, where *dest* may be a physical or DT neighbor. To meet the requirements of VPoD, each entry in MDT protocols used by VPoD is extended to a 6-tuple $< source, pred, succ, dest, cost, error >$, where *error* is the estimated position error of the *dest* node. If *dest* is a physical neighbor of u, *cost* is the link cost to *dest*. If *dest* is a multi-hop DT neighbor, *cost* is the routing cost to *dest*. In tuples where *dest* is neither a physical nor DT neighbor, both *cost* and *error* are empty.

We made another change to MDT join and maintenance protocols to accommodate VPoD as follows. In [11], nodes are globally identified by their coordinates which do not change. In VPoD, each node's virtual position is arbitrary and changes over time. Therefore, nodes in VPoD are identified by globally unique identifiers which are included in MDT messages. However, MDT protocols running under VPoD *do not require a location service* to provide a mapping from global identifiers to virtual positions. Nodes that are physical neighbors exchange messages and learn the updated virtual positions of each other. Also, during execution of the MDT protocols, whenever node u learns a new node x from node v, the message from v to u includes both the global identifier and virtual position of x. When the MDT protocols finish execution, each node knows the global identifiers and virtual positions of all of its physical and DT neighbors.

Additionally, during execution of the MDT join and maintenance protocols, every pair of DT neighbors exchange two messages, *Neighbor_Set_Request* and *Neighbor_Set_Reply*. Each of these messages carries its source node's position error and is also used to record the routing cost of the *reverse path* from its destination node to its source node. When the MDT protocols finish execution, every node knows the *cost* and *error* values of each of its DT neighbors. Also, a path from the node to each of its DT neighbors has been stored in forwarding tables of nodes along the path. The *error* values of physical neighbors that are not DT neighbors are exchanged by link-layer keep-alive messages.

Experimental results show that MDT protocols construct a correct multi-hop DT very quickly at system initialization. The protocols are highly resilient to churn, i.e., frequent and dynamic topology changes due to addition and deletion of nodes and links. They are also communication efficient because they do not use flooding to discover multi-hop DT neighbors [11].

D. Adjustment algorithm

During each execution of the adjustment algorithm (see pseudocode in Figure 6 with notation defined in Table I), a node u may change its position multiple times to find a position in the virtual space with less prediction error. Before algorithm execution, node u first computes its distances $\tilde{D}(u,v)$ to its physical and DT neighbors using their current virtual positions. (Note that its DT neighbor set and routing costs to DT neighbors do not change during algorithm execution.) Then,

Adjustment():

1. $e_{\text{sum}} \leftarrow 0$; // summed error of this adjustment, initialized to 0;
2. **for** all v in $P_u \cup N_u$ **do**
3. **if** ($v \in P_u$ **and** $\tilde{D}(u,v) > D(u,v)$) **or** $v \in N_u - P_u$ **then**
4. $t \leftarrow$ tuple in F_u such that $t.dest = v$;
5. $e_v \leftarrow t.error$;
6. $f \leftarrow e_u/(e_u + e_v)$; // confidence of this update
7. $x_u \leftarrow x_u + c_c \times f \times [D(u,v) - \tilde{D}(u,v)] \times \hat{u}(x_u - x_v)$;
   // $\hat{u}(x_u - x_v)$ is a unit vector in the direction of $x_u - x_v$
8. $e_{\text{sum}} \leftarrow e_{\text{sum}} + |D(u,v) - \tilde{D}(u,v)| / \tilde{D}(u,v)$;
   // add the error of this sample
9. **end if**
10. **end for**
11. $e_{\text{new}} \leftarrow e_{\text{sum}}/|P_u \cup N_u|$; // average error
12. $e_u \leftarrow e_u \times (1 - c_e) + e_{\text{new}} \times c_e$;
13. Send the updated x_u and e_u to all nodes in $P_u \cup N_u$;

Fig. 6. Pseudocode of the VPoD adjustment algorithm at node u

u updates its position (executes lines 4-7 of pseudocode) with respect to every multi-hop DT neighbor and some physical neighbors. Specifically, for a physical neighbor v, u updates its position with respect to v if u's distance to v is larger than u's routing cost to v, that is, $\tilde{D}(u,v) > D(u,v)$ (see line 3 of pseudocode). During execution of the adjustment algorithm, there is no message exchange between node u and other nodes. At the end of algorithm execution, node u sends its updated position and position error to all of its physical and DT neighbors.

When node u makes a position adjustment with respect to v, it moves its position in the direction of $[D(u,v) - \tilde{D}(u,v)] \times \hat{u}(x_u - x_v)$, where x_u and x_v are position vectors, and $\hat{u}(x_u - x_v)$ is a unit vector in the direction of $x_u - x_v$. The magnitude of the movement is proportional to the magnitude of $D(u,v) - \tilde{D}(u,v)$, where $D(u,v)$ is routing cost from u to v and $\tilde{D}(u,v)$ is distance between them. If $D(u,v) < \tilde{D}(u,v)$, u moves towards v; if $D(u,v) > \tilde{D}(u,v)$, u moves away from v. The magnitude of the movement is also proportional to the confidence value f of this adjustment computed as follows. If v has a large position error, the position error of v may propagate to u. To mitigate such error propagation, neighbors with large position errors should have less influence in position updates than those with small errors. Similar to Vivaldi, each node u maintains a local variable e_u for its estimated position error. The confidence value f of the adjustment is defined to be

$$f = \frac{e_u}{e_u + e_v}$$

The update rule for each neighbor v that causes a position change is:

$$x_u = x_u + c_c \times f \times [D(u,v) - \tilde{D}(u,v)] \times \hat{u}(x_u - x_v)$$

where c_c is a tuning parameter to be determined (see Section IV-D). The value of $D(u,v)$ is available to u in the *cost* field of the tuple in F_u whose *dest* field is v. Note that for a multi-hop DT neighbor v, the cost field does not always store the minimum routing cost from u to v, because the path in the

multi-hop DT may not be the shortest one. However, since the main goal of adjusting with a multi-hop DT neighbor v is to move u away from v, we found that an over-estimate of the routing cost works effectively (because if $D(u,v) > \tilde{D}(u,v)$, u moves away from v).

After updating its position, node u also needs to update its estimated position error. For each update caused by neighbor v, u computes the prediction error $\tilde{e}_v$ by

$$\tilde{e}_v = |D(u,v) - \tilde{D}(u,v)| / \tilde{D}(u,v)$$

If v does not cause an update, $\tilde{e}_v = 0$. After checking all neighbors, u computes the average over all of its physical and DT neighbors:

$$e_{new} = \sum \tilde{e}_v / |N_u \cup P_u|$$

The position error of node u is then updated by a moving average:

$$e_u = e_u \times (1 - c_e) + e_{new} \times c_e$$

where c_e is another tuning parameter in the range (0, 1). The initial value of e_u is 1. We use $c_e = 0.25$ in our experiments.

At the end of the adjustment algorithm, node u sends the updated values of x_u and e_u to all neighbors (physical and DT).

E. Adaptive adjustment timeout

The number of *Adjustment*() executions for node u during an adjustment period is determined by $\lceil \frac{T_a}{\Delta_u} \rceil$, where T_a is the duration of the adjustment period and Δ_u is the adjustment timeout period of node u. One challenge is the choice of a proper value of Δ_u at different stages of the virtual position construction process. At the beginning of an A period, using small timeouts can help nodes rapidly find approximate positions. When node positions are relatively stable, the positions should be refined slowly for them to converge. Also the multi-hop DT constructed in the previous J period needs to be updated after several *Adjustment*() executions. If *Adjustment*() is executed too frequently with an outdated multi-hop DT, node positions may oscillate and do not converge.

We use an adaptive timeout technique to achieve fast and accurate convergence. The initial timeout Δ_{u0} is set to a small value, e.g., 2 sec. After that, each node calculates the average position error of its physical and DT neighbors, denoted by $\bar{e}$. The timeout is then changed to

$$\Delta_u = \min(\Delta_{u0} / \bar{e}, T_a)$$

Note that position errors are initialized to 1 and will decrease with time. When the virtual positions converge and become relatively stable, $\bar{e}$ trends towards 0 and results in a large Δ_u. Experimental results for different values of timeout are presented in Section IV-B.

III. Greedy Distance Vector (GDV) Routing

In GDV_basic (see pseudocode in the left column of Figure 7), when node u has a packet to forward, it uses the virtual positions of its physical neighbors and the destination t to compute estimated routing costs. GDV_basic does not use MDT. Furthermore, GDV_basic does not assume the use of VPoD; virtual positions of nodes may be provided by any virtual positioning protocol that can effectively embed routing costs into a virtual space (like VPoD). For each physical neighbor $y \in P_u$, the estimated routing cost from u to t via y is $R_y = c(u,y) + \tilde{D}(y,t)$. Using GDV_basic, node u selects the physical neighbor v such that $R_v = \min_{y \in P_u} R_y$. To avoid routing loops, GDV requires that $\tilde{D}(v,t) < \tilde{D}(u,t)$, i.e., v is closer to the destination than u in the virtual space. If $R_v < \tilde{D}(u,t)$ is satisfied, then $\tilde{D}(v,t) < \tilde{D}(u,t)$ holds, and u sends the packet to v. Note that lines 1-3 in left column of Figure 7 perform DV routing with distance vectors computed locally rather than communicated and stored.

GDV_basic(u, t):	**GDV(u, t)**:
1. For each physical neighbor y, $R_y \leftarrow c(u,y) + \tilde{D}(y,t)$;	**1.** For each physical neighbor y, $R_y \leftarrow c(u,y) + \tilde{D}(y,t)$;
2. Let v be the physical neighbor that minimizes R_y;	**2.** For each multi-hop DT neighbor y, $R_y \leftarrow D(u,y) + \tilde{D}(y,t)$;
3. if $R_v < \tilde{D}(u,t)$ **then**	**3.** Let v be the neighbor that minimizes R_y;
send the packet to v;	**4. if** $R_v < \tilde{D}(u,t)$ **then**
4. else	send the packet to v directly or by the multi-hop path;
GR(u, t); // geographic routing	**5. else**
5. end if	MDT_greedy(u, t);
	6. end if

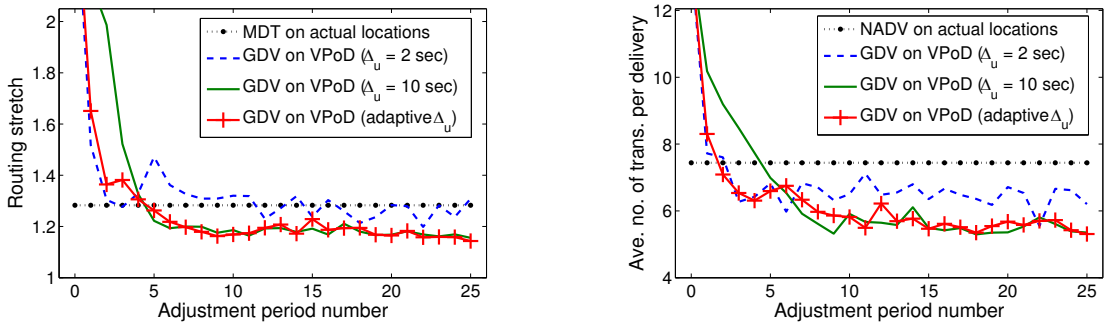
Fig. 7. GDV pseudocode at node u to destination t

In line 4 in left column of Figure 7, GDV_basic switches to a geographic routing protocol, GR, based upon greedy forwarding with some recovery method to move packets out of local minima. (Almost any existing geographic routing protocol may be used as GR.) GR uses the virtual positions of nodes for its forwarding decision without any consideration of link costs. When a node, say w, receives a packet that is in GR recovery, w skips lines 1-3 in the GDV_basic pseudocode and runs GR. (This detail is omitted in Figure 7.)

Since a multi-hop DT is already constructed by VPoD, the version of GDV we use in this paper (see pseudocode in the right column of Figure 7) also makes use of the set of multi-hop DT neighbors to improve routing performance and provide guaranteed delivery. We know that MDT forwarding in a correct multi-hop DT provides guaranteed delivery in d-dimensional spaces, $d \geq 2$, as well as a routing stretch close to 1.

Using GDV, when node u has a packet to forward, it uses the virtual positions of its physical and multi-hop DT neighbors and the destination t to compute estimated routing costs. VPoD provides u with the virtual position, routing cost, and a forwarding path to each of its multi-hop DT neighbors. For every multi-hop DT neighbor y, node u computes the estimated routing cost via y to t, $R_y = D(u,y) + \tilde{D}(y,t)$. Using GDV, u selects the node v such that $R_v = \min_{y \in P_u \cup N_u} R_y$. If $R_v < \tilde{D}(u,t)$, u sends the packet to v directly if v is a physical neighbor, or by the multi-hop path to v if v is a multi-hop DT neighbor (line 4 in right column of Figure 7).

If $R_v < \tilde{D}(u,t)$ is not satisfied, node u runs MDT-greedy using virtual positions of nodes without any consideration of



(a) metric is hop count

(b) metric is ETX

Fig. 8. Routing performance for different timeout values

routing costs (line 5 in right column of Figure 7). When a node, say w, receives a packet that is being forwarded in a virtual link and w is not the virtual link's destination, it skips lines 1-4 in the GDV pseudocode and runs MDT-greedy. (This detail is omitted in Figure 7.) Since executing line 4 in the GDV pseudocode strictly reduces a packet's distance to its destination in the virtual space, it is straightforward to prove that GDV provides guaranteed delivery because MDT-greedy provides guaranteed delivery.

A. GDV for different routing metrics

GDV can use any routing metric that DV uses, such as, hop count, latency, ETX, ETT, energy consumption, and propagation distance, etc. Both GDV and DV require a metric m that is positive and additive. The metric, however, may be *asymmetric*, namely, it is not required that $m(u,v) = m(v,u)$ for two physical neighbors, u and v. The following example illustrates GDV's requirement of additivity and non-requirement of symmetry. In MDT protocols, when node a sends a Neighbor_Set_Request message to node b along the path a-x-y-b, the message's routing cost field is initialized to zero at node a. Then node x adds $c(x,a)$ to the field. Later, node y adds $c(y,x)$ to the field. Finally, node b adds $c(b,y)$ to the field. The cumulative value provides node b, the destination of the message, its routing cost back to node a. Subsequently, node b sends a Neighbor_Set_Reply message to a along the reverse path and node a obtains from the message its routing cost to b. Note that the costs of b-a and a-b paths may be different.

When a routing metric captures more network and link characteristics (such as, link quality by ETX [4] and both link quality and capacity by ETT [6]) the metric can be used to provide higher throughput for shortest-path routing. GDV is a geographic routing protocol designed to take advantage of such routing metrics. We found that even when hop count is used as the routing metric, GDV has better routing stretch performance than prior geographic routing protocols. This is because the distance in virtual space is better than the geographic distance in physical space for predicting routing stretch.

IV. Performance Evaluation

A. Methodology

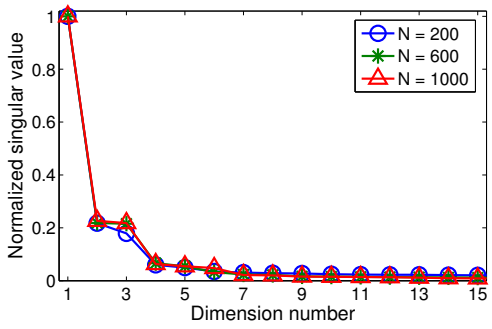
We evaluate the performance of GDV using a packet-level discrete-event simulator. Queuing delays are not simulated because we do not evaluate performance metrics that depend on congestion, e.g., end-to-end throughput and latency. Instead, random message delivery times from one node to the next are sampled from a uniform distribution over a specified time interval.

Performance criteria. GDV works for any routing metric that is positive and additive. For this paper, we used two common metrics in our experiments, namely, hop count and ETX. When using *hop count* as metric, we evaluate the routing stretch of each protocol. The *routing stretch* value between a pair of source and destination nodes is defined to be the ratio of the hop count in the selected route to the hop count in the shortest route in the connectivity graph. When using *ETX* as metric, we evaluate the average *number of transmissions* used to deliver a packet from a source node to a destination node. The routing stretch and number of transmissions shown in the figures are the average values over all source-destination pairs in the network. Using hop count as metric, we compare GDV with MDT-greedy. Using ETX as metric, we compare GDV with NADV [14].[5] To give advantage to NADV and MDT-greedy in the comparisons, we used *accurate node locations* for NADV and MDT-greedy in our experiments.[6]

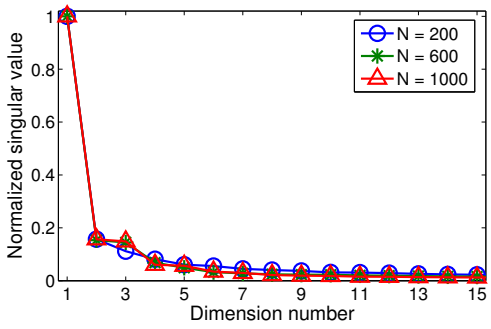
We measure the storage cost of a routing protocol by counting the number of distinct nodes a node needs to know (and store) to perform forwarding, and computing the average value over all nodes. This represents the storage cost of a node's minimum required knowledge of other nodes. It has been validated that the overall storage cost for forwarding is linearly proportional to the number of distinct nodes stored [11]. This metric, unlike counting bytes, requires no implementation assumptions which may cause bias when different routing protocols are compared.

[5] PRR×distance [20] is similar to NADV with PRR = 1/ETX.

[6] MDT-greedy provides guaranteed delivery for nodes with inaccurate or arbitrary coordinates. However, its routing stretch is lower when node locations are known more accurately.

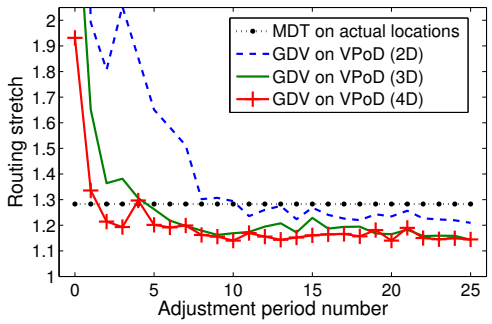


(a) metric is hop count

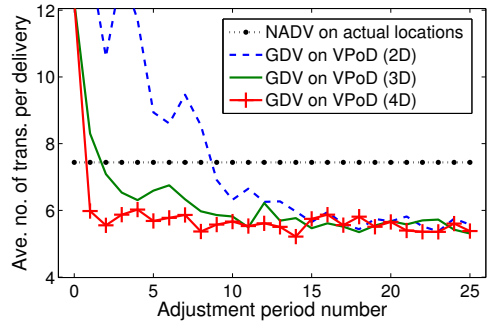


(b) metric is ETX

Fig. 9. Normalized singular values for different network sizes



(a) metric is hop count



(b) metric is ETX

Fig. 10. Routing performance for 2D, 3D, and 4D

Creating general connectivity graphs and ETX values. We used the link-layer simulator developed by the authors of [20] to create connectivity graphs and link costs (ETX values). Initially, N nodes are randomly placed in a 2D space. The packet reception rate (PRR) between two nodes is computed as a function of the distance, node density, and other parameters including path loss exponent, shadowing standard deviation, modulation and encoding schemes, output power, noise floor, preamble and frame lengths, and randomness. We use the default values for all parameters [20]. If the packet reception rate between two nodes is greater than 0.1, a physical link is placed between the two nodes in the connectivity graph. The ETX value of the link (in each direction) is the inverse of the PRR value.

For some experiments, we also randomly placed some large obstacles in the 2D space. Nodes are not placed in space occupied by obstacles. If the line between two nodes intersects any obstacle, there is no physical link between the nodes.

We will first present experimental results for 200-node networks. In section IV-G, the number of nodes is varied from 100 to 1000.

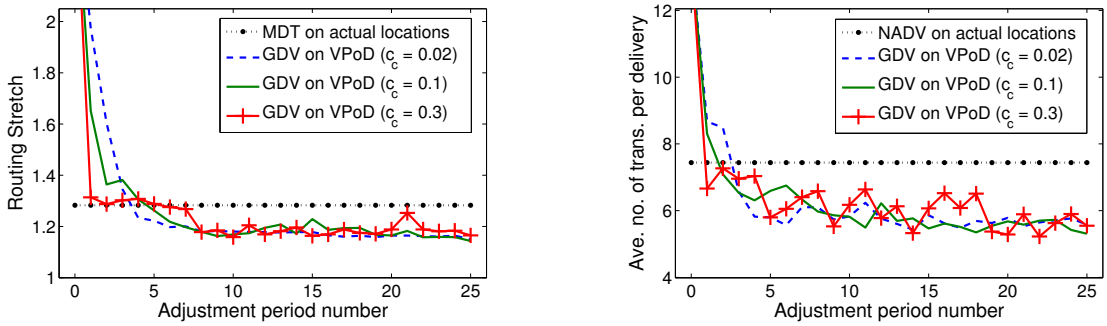
B. Adaptive adjustment timeout

We conducted many experiments for different values of adjustment timeout. We show representative results for a 200-node network in Figure 8. Nodes are in a 100m×100m 2D physical space. The average number of physical neighbors per node is 14.5. VPoD assigns node positions in a 3D virtual space. Routing performance versus adjustment period number (which represents time) is presented for hop count used as metric in Figure 8(a) and for ETX used as metric in Figure 8(b). The duration of an adjustment period is $T_a = 20$ seconds. Note that when the adjustment timeout is a small value (2 seconds), nodes can find their approximate positions after two periods. However, the routing performance keeps oscillating after that. On the other hand, using a large adjustment timeout (10 seconds) slows down the convergence. Adaptive timeout is the best strategy. Using adaptive timeout, the convergence is as fast as using a small timeout and the quality of virtual positions after convergence is similar to that from using a large timeout. We used adaptive timeout for all other experiments to be presented in this paper.

C. Choice of Dimensionality

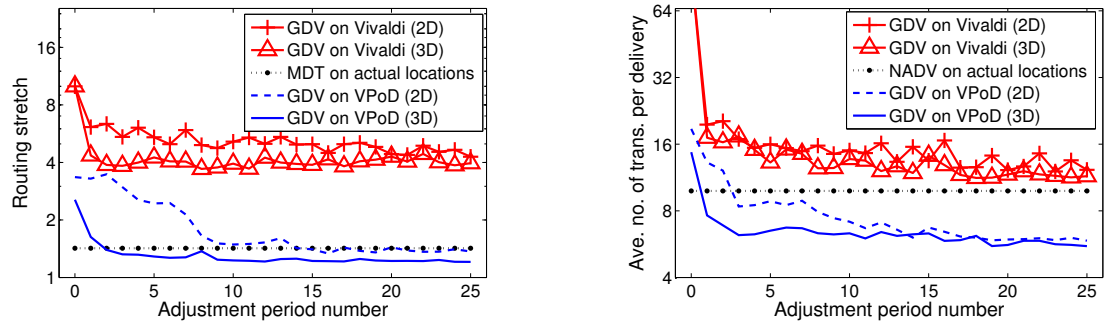
We use Principal Component Analysis (PCA) to determine whether a low-dimensional space can be used to effectively model routing costs of wireless networks. We then use it to find an appropriate dimensionality to use and we present experimental results to validate the PCA results.

PCA relies on Singular Value Decomposition (SVD). The input of SVD is an $N \times N$ matrix M, where each element m_{ij} is the routing cost from node i to node j. SVD factors M into the product of three matrices: $M = U \cdot S \cdot V^T$, where S is a diagonal matrix with nonnegative elements s_i. The diagonal elements are called *singular values* of M, which are ordered non-increasingly.



(a) metric is hop count (b) metric is ETX

Fig. 11. Routing performance for different values of tuning parameter c_c



(a) metric is hop count (b) metric is ETX

Fig. 12. Routing performance with four randomly placed obstacles

From $M = U \cdot S \cdot V^T$, we have $m_{ij} = \sum_{k=1}^{N} s_k u_{ik} v_{jk}$. If singular values $s_1, ..., s_d$ are much larger than the rest, we may approximate m_{ij} by $m_{ij} \approx \sum_{k=1}^{d} s_k u_{ik} v_{jk}$. This means that the routing cost matrix M can be embedded in a d-dimensional Euclidean space with low errors.

Figure 9 shows our experimental results for networks of 200, 600 and 1000 nodes. Each data point represents the average result from 20 different networks. The routing costs in the input matrix are measured in hop count for experiments in Figure 9(a), and in ETX for experiments in Figure 9(b). The singular values shown are normalized. The first three singular values are much larger than the remaining ones. Also as the network size increases, the third singular value increases in magnitude, which implies that the third dimension is more important for a larger network size.

We have performed many experiments for different networks embedded in 2D, 3D, and 4D virtual spaces. Figure 10 shows representative results of routing performance for 2D, 3D and 4D, using the same 200-node network for experiments in Figure 8. After 10 adjustment periods, the routing performance of GDV is better than MDT and NADV for all three virtual spaces. For 4D, the routing performance is close to the converged value after just one or two adjustment periods. 2D requires many more adjustment periods to converge. Note that the converged values of 4D are not much better than those of 3D. This observation is consistent with the PCA results in Figure 9.

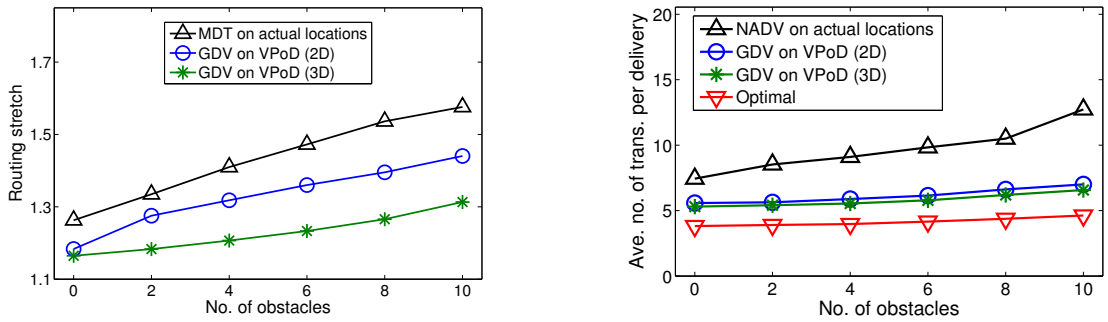
From the PCA and experimental results, 2D or 3D are good choices. As to be shown in Section IV-F, both the storage and communication costs of VPoD in 4D are significantly higher than those in 2D or 3D.

D. Impact of tuning parameter

The tuning parameter c_c controls the size of movement in position updates. We tried different values of c_c using the same network used for experiments shown in Figure 8. A 3D virtual space is used for VPoD. Figure 11 shows that a smaller value ($c_c = 0.02$) causes slower convergence in the first few adjustment periods but its convergence is still quite fast and accurate. When a large value ($c_c = 0.3$) is used, the convergence is fast at the beginning, but there are oscillations in the ETX experiments (see Figure 11(b)). VPoD with $c_c = 0.3$ still finds good virtual positions after 20 adjustment periods. Empirically, VPoD is quite robust to different values of c_c because VPoD uses two other adaptive values to control adjustments, i.e., confidence and timeout. We used $c_c = 0.1$ for all other experiments presented in this paper.

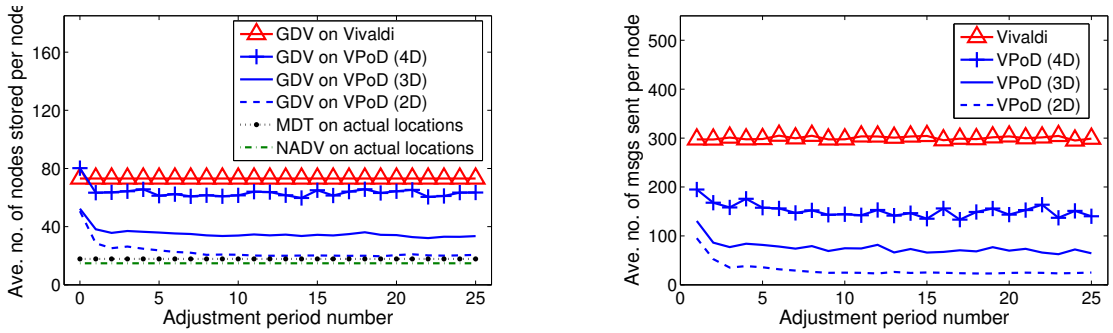
E. Impact of obstacles

The physical space of practical wireless networks may include large obstacles that block wireless transmissions. Thus we also evaluated GDV for networks with obstacles. In these experiments, each obstacle is a 10m×10m square. We randomly placed four obstacles in a 100m×100m physical space



(a) metric is hop count (b) metric is ETX

Fig. 13. Routing performance vs. number of obstacles



(a) Storage cost (b) control message cost in an adjustment period

Fig. 14. Storage cost and communication cost of VPoD

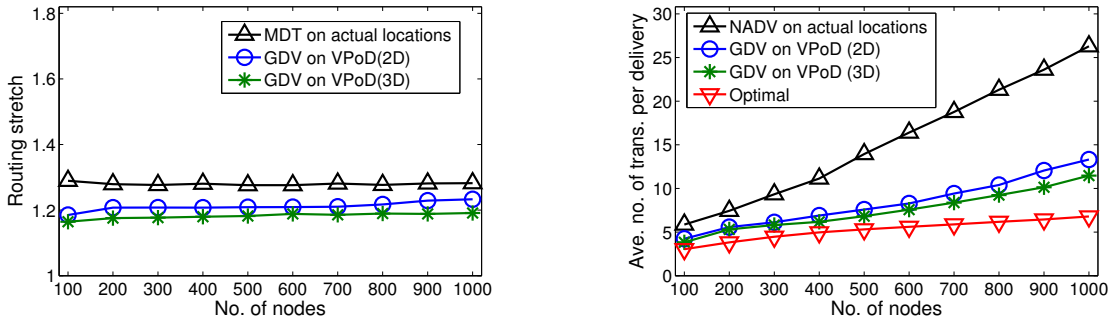
with the same 200-node network used for experiments in Figure 8. We also quantitatively compared VPoD with Vivaldi, by running GDV on virtual positions constructed by them. Similar to the experiments in Figure 2, we allow each node of Vivaldi to sample both one-hop and two-hop neighbors. The results are shown in Figure 12. GDV on VPoD outperforms both MDT and NADV on actual locations and it outperforms GDV on Vivaldi by a very large margin. (We found the performance of GDV on Vivaldi to be consistently poor. For the sake of clarity of presentation, we will omit results for GDV on Vivaldi in Figures 13, 15, and 16 to be presented.)

Next we varied the number of obstacles from 0 to 10 in the 100m$\times$100m physical space for 200-node networks. The results are shown in Figure 13. Each data point is the average value of 20 simulation runs for 20 different networks. For comparison, we also show the optimal values of shortest path routing using ETX as metric in Figure 13(b). In the same figure, the average number of transmissions of NADV increases from 7.44 (0 obstacle) to 12.73 (10 obstacles), while that of GDV on VPoD (3D) increases from 5.31 (0 obstacle) to 6.57 (10 obstacles). Note that the routing performance of GDV on VPoD is fairly close to that of optimal routing.

F. Storage and communication costs

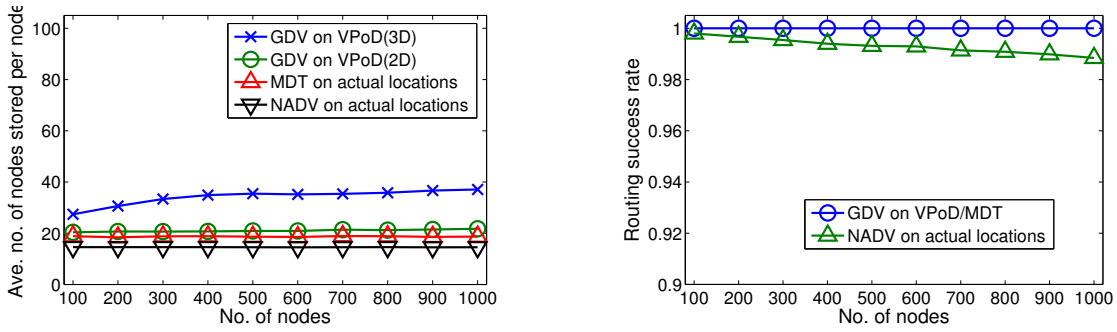
A multi-hop DT requires extra storage for multi-hop DT neighbors. The amount of extra storage varies during the course of the VPoD construction. At the beginning, most DT neighbors are not physical neighbors because the initial positions are fairly arbitrary. When VPoD has converged, the physical and DT neighbor sets have a large overlap. We evaluated both storage and communication costs using the same 200-node network for experiments in Figure 8. Figure 14(a) shows storage cost over time. The routing metric is hop count. (Results for the ETX metric are similar and not shown.) All three curves of GDV on VPoD start from high values and then drop after two adjustment periods. The storage cost of VPoD in 2D after convergence is very close to those of MDT and NADV on actual locations. The storage cost of VPoD in 4D after convergence is much higher than those in 2D and 3D, but still lower than that of two-hop Vivaldi. NADV requires each node to store physical neighbors only and has the lowest storage cost.

The average number of control messages sent per node in each adjustment period for constructing virtual positions is shown in Figure 14(b) for VPoD and Vivaldi. The message cost of VPoD includes both the multi-hop DT construction and adjustment update messages. The routing metric is hop count. (Results for the ETX metric are similar and not shown.) VPoD in 2D has the lowest message cost, which is about 20 messages per join-and-adjustment period. After convergence, the message costs of VPoD in 3D and 4D are about 60 and 140 messages, respectively, per join-and-adjustment period. Two-hop Vivaldi requires many more messages. We do not show message costs for MDT and NADV on actual locations. Given location information, they require a one-time construction with



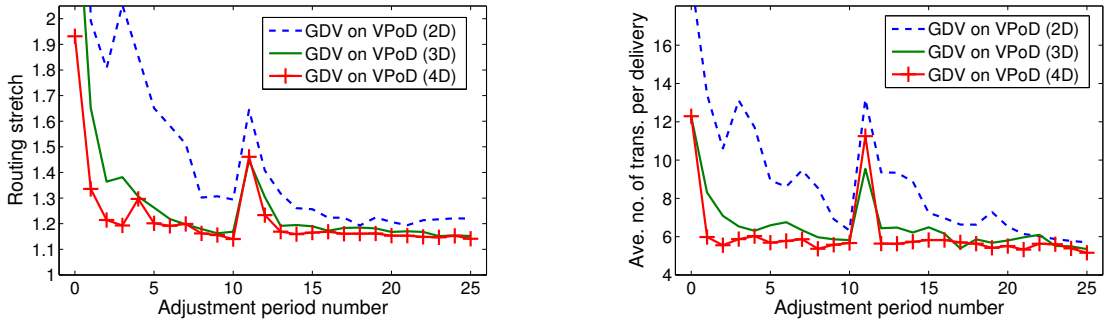
(a) metric is hop count (b) metric is ETX

Fig. 15. Routing performance versus N



(a) Storage cost (b) Routing success rate

Fig. 16. Storage cost and routing success rate versus N



(a) metric is hop count (b) metric is ETX

Fig. 17. Routing performance under churn

low message costs. But they require localization methods which have message and other costs to provide accurate location information.

G. Varying the number of nodes

We evaluate the performance of GDV for network size (N) from 100 to 1000 nodes. For 200-node experiments, the size of the physical space is 100m×100m. For a smaller (or larger) number of nodes, the size of the physical space is scaled down (or up) proportionally such that the average number of physical neighbors per node is kept at 14.5. No obstacles are placed. Each data point shown is the average value of 20 simulation runs for 20 different networks.

Figure 15(a) shows routing stretch versus N. GDV on VPoD performs better than MDT on actual locations. (Note that MDT has been shown to provide the lowest routing stretch when compared to other geographic routing protocols [11].) The routing stretch values of both GDV and MDT remain low as N increases.

Figure 15(b) shows that the average number of transmissions increases with N for all protocols (including optimal routing). NADV increases a lot more than GDV. For N =1000, the average number of transmissions of GDV is only half of that of NADV.

Figure 16(a) shows storage cost versus N. NADV has the lowest cost, followed in order by MDT, GDV on VPoD (2D), and GDV on VPoD (3D). The storage costs for all protocols remain low as N increases.

Figure 16(b) shows the routing success rates of different protocols. GDV and MDT both provide guaranteed delivery (the routing success rate was 100% in every experiment). The routing success rate of NADV is below 100% and decreases with N because NADV's recovery method from local minima does not work well for general connectivity graphs used in the experiments.

H. Resilience to dynamic topology changes

GDV and VPoD are highly resilient to dynamic network topology changes (*churn*) because VPoD uses MDT which is highly resilient. For the same 200-node network used for the experiments in Figure 10, we introduced node churn after the 10th adjustment period and at the beginning of the 11th join period. At the same time, 150 nodes (out of 200 nodes) failed and 150 new nodes joined. Each failed node became silent. Each new node chose its position in the virtual space to be the center of the positions of its physical neighbors that have a position error less than 1. Its own position error is set to 1.

Figure 17 shows the routing performance of GDV using VPoD in 2D, 3D, and 4D. Note that the routing performance for each routing metric (hop count or ETX) becomes worse immediately after churn. However, routing performance quickly converges to a low value after several adjustment periods (just 2-3 periods for 3D). The routing performance after 20 periods in total is as good as the performance shown in Figure 10 for experiments with a static topology. These and similar results from other churn experiments show that GDV and VPoD are very resilient to dynamic topology changes.

V. Conclusion

GDV is the *first* geographic routing protocol designed to optimize end-to-end path costs using *any additive routing metric*, such as, latency, ETX, and ETT which capture network and link characteristics. GDV provides guaranteed delivery and much better routing performance than existing geographic routing protocols using accurate location information. As a geographic protocol, GDV's storage cost per node remains low as network size increases.

GDV uses virtual positions of nodes provided by a new virtual positioning protocol, VPoD, which assumes that each node can measure its routing costs to directly-connected neighbors only. GDV and VPoD are designed for wireless networks without location information. Therefore, no localization protocol is needed. Unlike prior virtual positioning systems designed for hosts with Internet routing support (e.g., Vivaldi and GNP), VPoD does not require routing cost measurements to distant nodes or landmarks. VPoD is also communication efficient because it does not use flooding.

Acknowledgment

This research is sponsored by National Science Foundation grant CNS-0830939.

References

[1] P. Bose and P. Morin. Online routing in triangulations. *SIAM journal on computing*, 33(4):937–951, 2004.

[2] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. In *Proc. of the International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, 1999.

[3] A. Caruso, S. Chessa, S. De, and R. Urpi. GPS Free Coordinate Assignment and Routing in Wireless Sensor Networks. In *Proceedings of IEEE INFOCOM*, pages 150–160, 2005.

[4] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proceedings of ACM MobiCom*, 2003.

[5] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings ACM SIGCOMM*, 2004.

[6] R. Draves, J. Padhye, and B. Zill. Routing in Multi-radio, Multi-hop Wireless Mesh Networks. In *Proceedings of ACM Mobicom*, 2004.

[7] 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 Sensor Networks. In *Proc. of NSDI*, 2005.

[8] S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, second edition, 2004.

[9] B. Karp and H. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of ACM Mobicom*, 2000.

[10] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *Proceedings of USENIX NSDI*, 2005.

[11] S. S. Lam and C. Qian. Geographic Routing in d-dimensional Spaces with Guaranteed Delivery and Low Stretch. In *Proceedings of ACM SIGMETRICS*, June 2011.

[12] D.-Y. Lee and S. S. Lam. Protocol design for dynamic Delaunay triangulation. Technical Report TR-06-48, The Univ. of Texas at Austin, Dept. of Computer Science, December 2006 (an abbreviated version in *Proceedings IEEE ICDCS*, June 2007).

[13] D.-Y. Lee and S. S. Lam. Efficient and Accurate Protocols for Distributed Delaunay Triangulation under Churn. In *Proceedings of IEEE ICNP*, November 2008.

[14] S. Lee, B. Bhattacharjee, and S. Banerjee. Efficient Geographic Routing in Multihop Wireless Networks. In *Proceedings of ACM MobiHoc*, 2005.

[15] B. Leong, B. Liskov, and R. Morris. Geographic Routing without Planarization. In *Proceedings of USENIX NSDI*, 2006.

[16] B. Leong, B. Liskov, and R. Morris. Greedy Virtual Coordinates for Geographic Routing. In *Proceedings of IEEE ICNP*, 2007.

[17] Y. Liu, L. M. Ni, and M. Li. A Geography-free Routing Protocol for Wireless Sensor Networks. In *Proceedings of HPSR*, 2005.

[18] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proceedings of INFOCOM*, 2002.

[19] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing without Location Information. In *Proceedings of ACM Mobicom*, 2003.

[20] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proceedings of ACM SenSys*, 2004.

[21] M.-J. Tsai, H.-Y. Yang, B.-H. Liu, and W.-Q. Huang. Virtual-Coordinate-Based Delivery-Guaranteed Routing Protocol in Wireless Sensor Networks. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 17, 2009.

[22] Y. Zhao, Y. Chen, B. Li, and Q. Zhang. Hop ID: A Virtual Coordinate based Routing for Sparse Mobile Ad Hoc Networks. *IEEE Transaction on Mobile Computing*, 2007.