Greedy Routing by Network Distance Embedding

Chen Qian, *Member, IEEE, Member, ACM* and Simon S. Lam, *Fellow, IEEE, Fellow, ACM*

***Abstract*—Greedy routing has been applied to both wireline and wireless networks due to its scalability of routing state and resiliency to network dynamics. In this work, we solve a fundamental problem in applying greedy routing to networks with arbitrary topologies, i.e., how to construct node coordinates such that greedy routing can find near-optimal routing paths for various routing metrics. We propose Greedy Distance Vector (GDV), the first greedy 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 physical location information. Instead, it relies on a novel virtual positioning protocol, VPoD, which provides *network distance embedding*. Using VPoD each node assigns itself a position in a virtual space such that the Euclidean distance between any two nodes in the virtual space is a good estimate of the routing cost between them. Experimental results using both real and synthetic network topologies 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 greedy routing protocol, the routing state of GDV per node remains small as network size increases. We also show that GDV and VPoD are highly resilient to dynamic topology changes.**

I. Introduction

Greedy routing protocols have been proposed for both large-scale wireless [2] [10] [11] [15] [12] and wireline layer-2 networks [1] [27] [23] [32] [5]. In greedy routing, the routing state needed per node is independent of network size; hence greedy routing provides scalability of routing state as well as resiliency to network dynamics. Greedy routing is also known as geographic routing because most prior studies use the geographic locations of nodes for routing decisions. Geographic routing uses *greedy forwarding* as its basis, i.e., for a packet with destination t, a node u selects, as the next hop to t, a physical (one-hop) neighbor that minimizes the physical distance from a physical neighbor to t among all of u's physical neighbors.

Geographic routing finds good routing paths only if the physical distance between two nodes can, at least approximately, predict the routing cost between them. This assumption is invalid in wireline networks. For example, two routers in the same room may connect to different networks. Even for many wireless networks, the physical distance between two nodes is a poor predictor of the routing cost between them. We illustrate this point by analyzing the trace data of GreenOrbs [17], a large-scale wireless sensor network project deployed in Tianmu Mountain, China. We focus on two different network deployments named "Network1" and "Network2", with 200 and 213 TelosB motes respectively based on the measurement results of GreenOrbs on 8/3/2011 and 8/5/2011. When two nodes can receive packets from each other and the RSSI is higher than a threshold (-80 dBm), we consider that there exists a communication link between them.

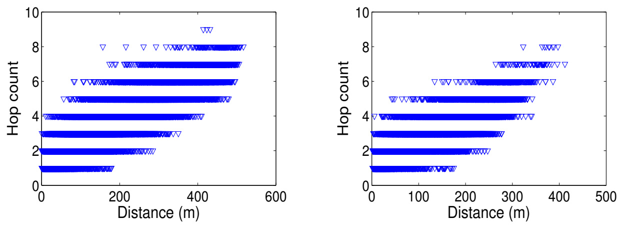


Fig. 1. Physical distances and network distances (in hop count) for all pairs of nodes in two GreenOrbs networks

Figure 1 shows the physical distances and shortest-path hop counts for all pairs of nodes in Network1 and Network2. Physical distances are computed based on GPS readings. From Figure 1, note that the physical distance between two nodes is not an effective estimator of the network distance between them in hop count. Suppose node a has two neighbors b and c which are 150 m and 130 m away from the destination respectively. a will send the message to c which is closer to the destination. However, from the results in Figure 1, it is possible that c is 5 hops to the destination but b is only one hop to the destination. As a result, geographic routing using physical locations of nodes would make suboptimal routing decisions and result in end-to-end paths with large routing costs.

The problem becomes more challenging if we want to incorporate various link cost metrics into geographic routing, such as, latency, ETX [4], and ETT [7], as well as metrics reflecting available bandwidth [31]. These metrics are used to incorporate the effects of link loss, capacity, asymmetry, and interference and to achieve high throughput. However, they were intended for shortest-path routing protocols and cannot be used by geographic routing directly.

In this paper, we present *Greedy Distance Vector* (GDV), the first greedy routing protocol designed with the objective of providing near-optimal paths for *any additive routing metric*. GDV is designed for layer-2 networks (wireline and wireless) that do not use IP routing. To apply GDV, each node computes a virtual position (position in a virtual space) for itself by running the *Virtual Position by Delaunay* (VPoD) protocol to be presented in this paper. VPoD provides the property of *network distance embedding*, i.e., the Euclidean distance between each pair of nodes in the virtual space is a good estimate of the routing cost between them.

This work was sponsored by National Science Foundation grants CNS-0830939, CNS-1214239, and CNS-1464335, and University of Kentucky College of Engineering Startup Grant. An abbreviated version of this paper [22] appeared in Proceedings of IEEE ICDCS Conference, Minneapolis, MN, June 2011.

Chen Qian is with the Department of Computer Science, University of Kentucky, Lexington, KY, 40506 (e-mail: qian@cs.uky.edu). Simon S. Lam is with the Department of Computer Science, The University of Texas at Austin, Austin, TX 78712 (e-mail: lam@cs.utexas.edu).

In GDV routing, a node u chooses a neighbor v as the next hop to destination t to minimize the routing cost $c(u,x)+\tilde{D}(x,t)$ for $x \in N_u$ where $c(u,x)$ is the cost of link u-x and N_u is the neighbor set.[1] This routing decision process is similar to the well-known Distance Vector (DV) routing, in which a next hop node is chosen when it minimizes $c(u,x)+D(x,t)$ for $x \in N_u$ and $D(x,t)$ is from the distance vector. However, unlike DV routing, the routing cost $\tilde{D}(x,t)$ from x to t is *computed locally* by node u from the virtual positions of x and t. 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. Furthermore, as a greedy routing protocol, GDV does not have the disadvantages of DV routing, i.e., large routing state and slow convergence, which are impediments to scalability.

The GDV and VPoD protocols presented in this paper make use of a georgraphic routing protocol, MDT [12], which provides guaranteed delivery for nodes with arbitrary coordinates in a Euclidean space. The contributions of this paper include the following:

- GDV is the *first* greedy routing protocol designed to optimize end-to-end path costs using any *additive routing metric*, such as, routing metrics that capture network and link characteristics other than physical distances.
- GDV and VPoD are designed for layer-2 wireline and wireless networks without location information. Therefore, no localization protocol is needed.
- As a greedy routing protocol, GDV's storage cost per node remains low as network size (N) increases. The routing costs from neighbors to a destination are computed locally using virtual positions. (Unlike DV, there is no need for nodes to exchange distance-vector messages of size $O(N)$.)
- VPoD provides effective network distance embedding. GDV performs better than prior greedy routing protocols when hop count is used as metric and much better when ETX is used as metric.
- GDV provides guaranteed delivery when the network topology is static. GDV and VPoD are highly resilient to dynamic topology changes.
- Every node runs the same protocols. GDV and VPoD do not require special nodes, such as, beacons and landmarks, and do not use flooding.

GDV routing has been applied in the design and evaluation of a large-scale layer-2 network architecture, named ROME [21], [23], which is backwards compatible with Ethernet hosts. ROME protocols include a stateless multicast protocol, a Delaunay DHT, and host/service discovery protocols, all of which make use of GDV routing. In this paper, we present the ideas, design, specification, and performance evaluation of GDV routing.

The balance of this paper is organized as follows. In Section II, we present related work. In Section III, we introduce two topics that underlie the GDV design, namely: network distance embedding and multi-hop Delaunay triangulation (MDT). In Section IV, we present the VPoD protocol. In Section V, we present the GDV protocol. In Sections VI and VII, we present experimental results to evaluate VPoD and GDV performance on wireless as well as wireline network topologies. We show that GDV and VPoD are highly resilient to dynamic topology changes. In Section VII, we present two optimization techniques to make VPoD and GDV run more efficiently on network topologies that have a significant number of low-degree nodes. We conclude in Section VIII.

[1]In GDV, the neighbor set is actually extended to include a set of Delaunay triangulation neighbors to be introduced in Section III.

II. Related Work

Geographic routing protocols have been proposed for large-scale wireless networks, because the routing state required is independent of network size. Various schemes have been designed to move packets out of local minima. For 2D networks, GFG [2], GPSR [10], and their variants [11] use face routing on a planar graph constructed by planarization algorithms. Leong *et al.* [15] proposed using a spanning tree to guarantee delivery in 2D without planarization. Lam and Qian proposed MDT [12] which provides guaranteed delivery and low stretch in 3D as well as 2D, for any connected graph and node locations specified by accurate, inaccurate or arbitrary coordinates.

Virtual coordinate schemes have been proposed in the absence of geographic locations, such as NoGeo [24], BVR [8], VCap [3], HopID [33], GSpring [16], ABVCup [29], and PSVC [34]. None of them attempt to predict routing cost. Instead, their main purpose is to improve the packet delivery rate.

Traditional geographic routing protocols use hop count as the routing metric. De Couto *et al.* [4] showed that the hop count metric has poor throughput for routing in multi-hop wireless networks. Instead, several high-throughput routing metrics have been proposed, such as ETX [4] and ETT [7], which incorporate the effects of link loss, asymmetry, capacity and interference. Lee *et al.* [14] proposed normalized advance (NADV) for geographic routing. NADV selects the next-hop node that minimizes $\frac{ADV}{Cost}$, where ADV is the amount of decrease in geographic distance and $Cost$ is the link cost such as ETX. A similar method is presented by Seada *et al.* [26]. Both methods made geographic routing cost-aware of the next hop and more efficient. But, unlike GDV, they provide no routing cost information for the entire path.

The idea of latency (network distance) embedding in a virtual space was used by several Internet virtual positioning systems, such as, GNP [20] and Vivaldi [6]. They were designed for hosts with Internet routing support. Both protocols use latency as routing cost. More specifically, GNP requires that each node makes RTT measurements to a set of geographically distributed *landmark* nodes. Vivaldi requires that each node receives a significant fraction of its latency measurements from high-latency, geographically distributed nodes.

III. Background

A. Network distance embedding

To illustrate the point that Vivaldi requires measurements to nodes with high routing costs, consider the 121-node grid

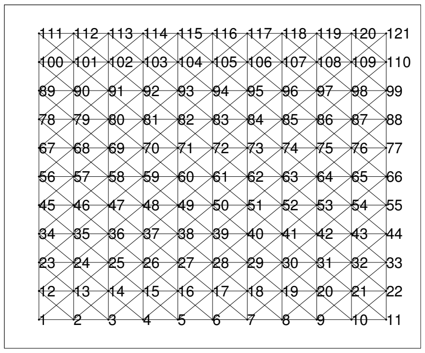
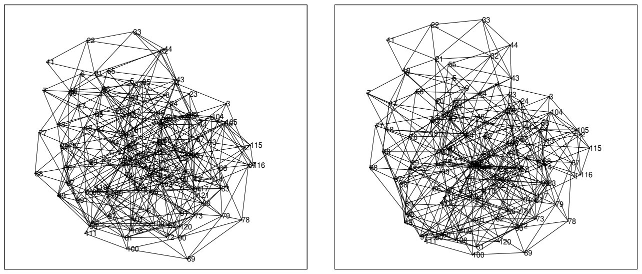


Fig. 2. 121-node network in 2D physical space



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

Fig. 3. Virtual positions constructed by 2-hop Vivaldi

network shown in Figure 2. Each node is only aware of its local connectivity and has no location information. We enhance the Vivaldi algorithm [6] 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 3 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 that are needed for virtual positions to predict routing costs accurately [6]:

- *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.

B. MDT routing support for VPoD

Since VPoD is designed for layer-2 networks that do not use IP routing, it uses a greedy routing protocol instead. VPoD has three requirements for such a greedy routing protocol: (i) The protocol allows each node to choose and adjust its location in a virtual space. (ii) The protocol provides guaranteed delivery. (iii) The protocol provides support for nodes running VPoD to converge to locations in the virtual space that satisfy the network distance embedding property.

Before presenting MDT routing, we 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 [9]. 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 denoted by $DT(S)$.

Nodes can construct a correct *distributed DT* by running an iterative search protocol to find their DT neighbors under the assumption that each node can directly communicate with every other node [13]. For multi-hop layer-2 networks, MDT protocols were designed for nodes to construct a distributed multi-hop DT with the following properties [12]: (i) Each node knows all of its physical neighbors and DT neighbors. (ii) Each node, say u, maintains a soft-state forwarding table, F_u. The forwarding tables of all nodes provide a forwarding path (virtual link) from every node to each of its multi-hop DT neighbors.

MDT-greedy routing: For a packet with destination t being forwarded by node u, ***if*** u is not a local minimum ***then*** the packet is forwarded to a physical neighbor of u closest to t; ***else***, the packet is forwarded, via a virtual link, to a multi-hop DT neighbor closest to t.

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 a Euclidean space (2D, 3D, or a higher dimension).

From the above description, MDT-greedy routing satisfies the first two requirements of VPoD. As for the third requirement, we will show that nodes running VPoD can quickly find locations that satisfy the network distance embedding property.

IV. Virtual Position Construction

We next present the VPoD protocol for nodes to find positions in a virtual space with the network distance embedding property. Initially, each node only knows its physical (one-hop) neighbors and the link costs to them. The link cost metric can be any one that is additive (e.g., hop count, latency, ETX, and ETT). Distances in the virtual space and routing costs are measured in the same units. Thus comparison, addition, and subtraction can be operated directly on distances and routing costs. Hereafter, when we say *distance*, we refer to the Euclidean distance between two nodes in the virtual space rather than the physical distance between them.

A. Main ideas of VPoD

A node boots up and assigns itself an initial location in a pre-specified virtual space (e.g., a rectangle in 2D). The node discovers its physical neighbors and exchanges ID and location information with them. When the node receives a start token to run VPoD, it forwards the token to all of its physical neighbors

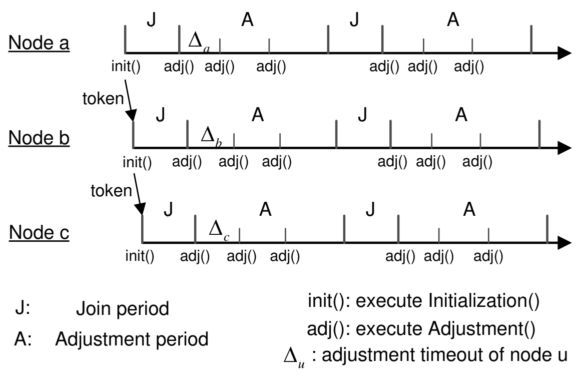

Node a
Node b
Node c
token
J
A
init()
adj()
J: Join period
A: Adjustment period
init(): execute Initialization()
adj(): execute Adjustment()
Δu : adjustment timeout of node u

Fig. 4. Main structure of VPoD

from which it has not received the same token. Any duplicate token received by a node is discarded.[2]

The following algorithm is used for initial location assignment:

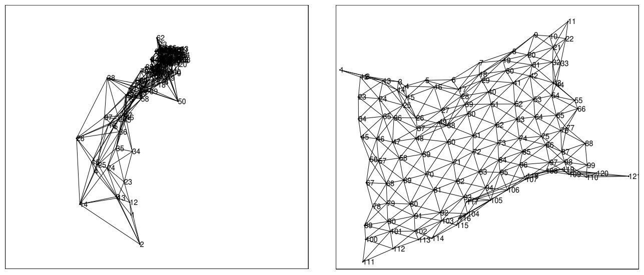
- If u is the starting node, u sets its position to the origin. Otherwise, at least one physical neighbor of u has initialized its position, namely, the token's sender.
- If only one physical neighbor, say v, of u has initialized its position, u sets its position at a random position on the circle or 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 in the disk centered at the mid-point whose radius is 1/10 of the distance between the two nodes.

All nodes that have received tokens run VPoD by first running MDT protocols to construct a multi-hop DT using their locations in the virtual space. MDT protocols have been modified to record routing costs from each node to its multi-hop DT neighbors. Each node then iteratively adjusts its position in the virtual space to reduce prediction errors of the distances between the node and its physical and multi-hop DT neighbors. For a node u, VPoD provides two types of adjustments:

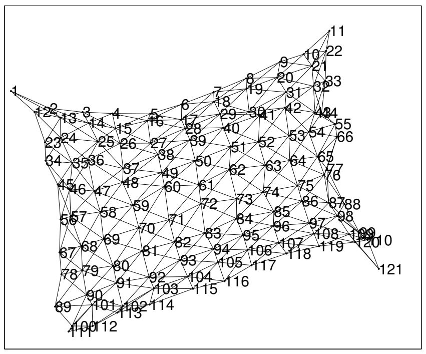
1) Adjustments with physical neighbors to preserve local relationships: If its distance to a physical neighbor v is larger than its link cost to v, u adjusts its position so that its distance to v is smaller.
2) Adjustments with DT neighbors to preserve global relationships: If its distance to a multi-hop 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. After receiving a token, each node runs MDT protocols during a period of time, called *J* period. The MDT protocols construct a multi-hop DT of a set of nodes in a distributed manner. Different nodes may join the multi-hop DT asynchronously

[2]The first node to receive a token may be predetermined by the network operator or by a leader election protocol based upon a simple criterion, such as, largest ID.



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



(c) After 20 adjustment periods

Fig. 5. Virtual positions constructed by VPoD

and when all nodes finish joining the correct multi-hop DT has been constructed [12]. 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 running MDT protocols in a *J* period and the adjustment algorithm in an *A* period. The MDT protocols and adjustment algorithm will be described in more detail in the sections to follow.

Note that after receiving a token, each node runs asynchronously. Different nodes may start their J and A periods at slightly different times. 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 will resume when there is node churn. The node that detects a change will initiate a token and pass it in the network to resume VPoD, similar to the initialization phase of VPoD. VPoD does not require any landmark or perimeter node and uses no flooding. Every node in the network runs the same VPoD protocol. The duration of a J period depends on the time for executing the MDT join protocol. When MDT join finishes, a J period stops. The length of an A period depends on message delivery times. For our experiments on wireless networks, the duration of an

```
Ad ustment():
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 $v \in N_u \quad P_u$ then
4.      $t \leftarrow$ tuple in $F_u$ such that $t.dest \quad v$;
5.      $e_v \leftarrow t.error$;
6.      $f \leftarrow e_u/(e_u \quad e_v)$;  // confidence of this update
7.      $x_u \leftarrow x_u + c_c \times f \times [D(u,v) \quad \tilde{D}(u,v)] \times \hat{u}(x_u - x_v)$;
                 // $\hat{u}(x_u - x_v)$ is a unit vector in the direction of $x_u$ - $x_v$
8.      $e_{\text{sum}} \leftarrow e_{\text{sum}} \quad |D(u,v) \quad \tilde{D}(u,v)| / \tilde{D}(u,v)$;
                                  // 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. $e_u \leftarrow e_u \times (1 \quad c_e) + e_{\text{new}} \times c_e$;
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

A period was set to 20 seconds. If the A period is too short, nodes may not able to receive enough information to adjust their coordinates. If it is too long, the convergence will be delayed.

We ran VPoD for the 121-node grid network in Figure 2. 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 2 where nodes are numbered. Note that adjustment periods for VPoD and 2-hop Vivaldi are defined differently. Experimental results in Figure 16 (to be presented) show that 2-hop Vivaldi uses much more storage and communication costs per adjustment period than VPoD.

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
$\tilde{D}(v,t)$	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 interval value of node u
c_c, c_e	tuning parameters to control the amounts of change in node position and position error

B. MDT extensions to support VPoD

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.

Additionally, during execution of the MDT 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 [12] 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 nodes use an efficient iterative search to find multi-hop DT neighbors (without flooding).

C. 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. Then, 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). 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. Note that u and v being physical neighbors usually implies a short network distance. Hence when their virtual distance is large, the protocol should move u towards v. When their virtual distance is always short, the protocol should allow them some range of movement and not force them to keep the exact distance.

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 VI-E). 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:

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

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 physical and DT neighbors.

D. Adaptive adjustment interval

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 interval of node u, controlled by a timeout timer. 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 intervals 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 interval technique to achieve fast and accurate convergence. The initial interval 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 interval 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 the adjustment interval are presented in Section VI-C.

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

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

V. GDV Routing

Using GDV, each node performs greedy forwarding in the multi-hop DT constructed by VPoD. 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, $\tilde{D}(y,t)$. For each physical neighbor y, node u computes the estimated routing cost via y to t by $R_y = c(u,y) + \tilde{D}(y,t)$ (line 1 in Figure 7). For every multi-hop DT neighbor y, node u computes the estimated routing cost via y to t by $R_y = D(u,y) + \tilde{D}(y,t)$ (line 2 in Figure 7).

Node u selects the node v such that $R_v = \min_{y \in P_u \cup N_u} R_y$ (line 3 in Figure 7). If $R_v < \tilde{D}(u,t)$, u sends the packet to v directly if v is a physical neighbor or by the virtual link to v if v is a multi-hop DT neighbor (line 4 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 routing costs (line 5 in 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.

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.

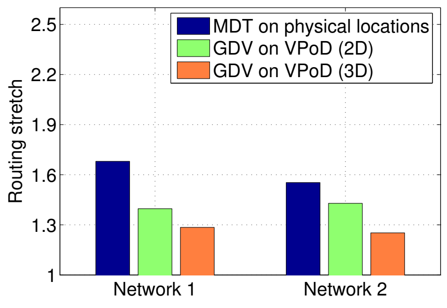


Fig. 8. Routing performance of GDV and MDT-greedy on GreenOrbs

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 [7]) the metric can be used to provide higher throughput for shortest-path routing. GDV is a greedy 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 cost in hop count.

Note that a location service is necessary for the sender to know the destination coordinates before executing GDV routing. We have designed a location lookup service for GDV, presented in another paper [21]. The data packet header of GDV requires space to store the coordinates and node ID of the destination. GDV uses 4 bytes per dimension for storing destination coordinates and 2 bytes for its ID. Hence for a 3D virtual space, the packet header requires 14 bytes. It is known that the EEE 802.15.4-compliant CC2420 radio used by many sensor nodes supports packets up to 127 bytes [34]. Hence there is sufficient remaining space to store data.

VI. Evaluation on Wireless Networks

A. Methodology

We evaluate the performance of GDV for both real GreenOrbs network topologies and synthetic wireless topologies generated by a packet-level discrete-event simulator [26]. In Section VII we will evaluate its performance on wireline network topologies. 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 another are sampled from a uniform distribution over a specified time interval $[10ms, 20ms]$.

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 the 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 the 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 the metric, we compare GDV with MDT-greedy. Using ETX as the metric, we compare GDV with NADV [14]. To give an advantage to NADV and MDT-greedy in the comparisons, we used *accurate node locations* for NADV and MDT-greedy in our experiments. We also compare GDV with PSVC [34], a recently proposed virtual coordinate system for wireless networks. Similar to most virtual coordinate protocols for wireless networks [3], [8], [16], [18], [24], [25], [29], [30], [33], PSVC only deals with hop count and cannot embed link costs into network distances.

MDT-greedy is used as the representative of geographic routing protocols because it has been shown [12] to provide the lowest routing stretch, for nodes with accurate or inaccurate coordinates, when compared to several well-known geographic protocols, i.e., GPSR running on GG, RNG, and CLDP graphs [2], [10], [11] and GDSTR [15].

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 [12]. This metric, unlike counting bytes, requires no implementation assumptions which may cause bias when different routing protocols are compared.

Creating general connectivity graphs and ETX values. We used the link-layer simulator developed by the authors of [26] 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 used the default values for all parameters [26]. 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. This threshold is set to 0.33 in experiments for sparse networks. 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

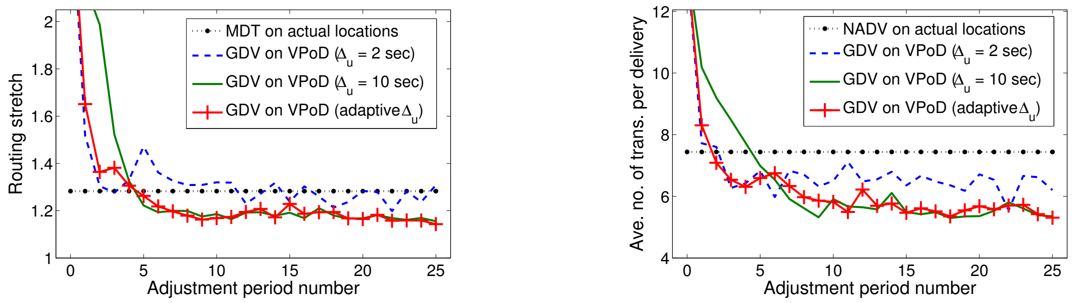


Fig. 9. Routing performance for different values of the adjustment interval: **(a)** metric is hop count, **(b)** metric is ETX.

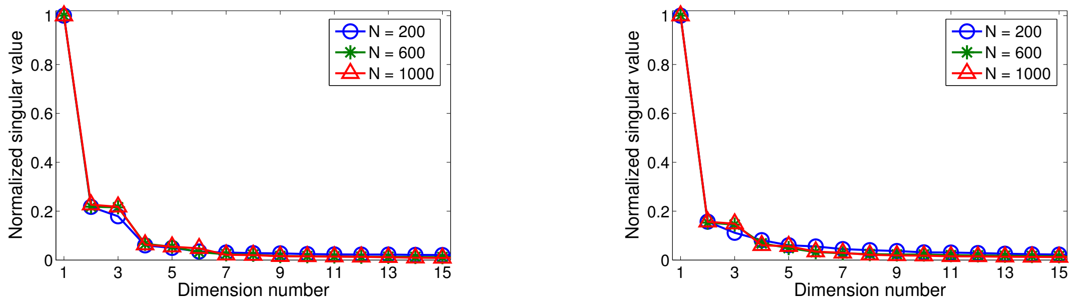


Fig. 10. Normalized singular values for different network sizes: **(a)** metric is hop count, **(b)** metric is ETX.

occupied by obstacles. Also if the line between two nodes intersects any obstacle, there is no physical link between the nodes.

B. GreenOrbs results

We first compare GDV with MDT-greedy for the two GreenOrbs networks whose average node degrees are 9.2 and 8.9. MDT-greedy uses physical locations of sensor nodes obtained by GPS. GDV uses no location information. In Figure 8, the routing stretch of MDT-greedy on physical locations is 1.6802 for Network1. The routing stretch is improved by GDV to 1.3965 on VPoD in 2D, and to 1.2847 on VPoD in 3D. For Network2 the routing stretch of MDT-greedy on physical locations is 1.5534. The routing stretch is improved by GDV to 1.4291 on VPoD in 2D, and to 1.2520 on VPoD in 3D. Note that GDV in 3D provides better routing stretch than GDV in 2D. We will discuss the choice of dimensionality in more detail in Section VI-D. In this set of experiments, VPoD stops after 20 J periods and 20 A periods.

C. Adaptive adjustment interval

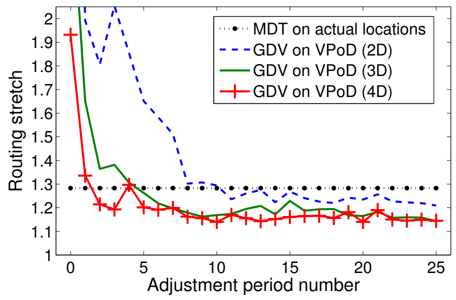
We conducted many experiments for different values of adjustment interval. We show representative results for a synthetic 200-node network in Figure 9. Nodes are in a 100m$\times$100m 2D physical space. The average number of physical neighbors per node is 14.5. VPoD assigns node positions in a 3D virtual space. Routing performance versus adjustment period number (which represents time) is presented for hop count used as the metric in Figure 9(a) and for ETX used as the metric in Figure 9(b). The duration of an adjustment period is $T_a = 20$ seconds. Note that when the adjustment interval 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 interval (10 seconds) slows down the convergence. Adaptive interval is the best strategy. Using adaptive interval, the convergence is as fast as using a small interval and the quality of virtual positions after convergence is similar to that from using a large interval. We used adaptive interval for all other experiments to be presented in this paper.

D. 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 multi-hop 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.

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.



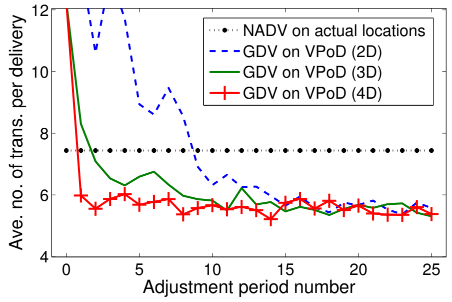


Fig. 11. Routing performance for 2D, 3D, and 4D: **(a)** metric is hop count, **(b)** metric is ETX.

Figure 10 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 10(a), and in ETX for experiments in Figure 10(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 11 shows representative results of routing performance for 2D, 3D and 4D, using the same 200-node network for experiments in Figure 9. After 10 adjustment periods, the routing performance of GDV is better than MDT-greedy 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 10.

From the PCA and experimental results, 2D or 3D are good choices. This is because both the storage and communication costs of VPoD in 4D are significantly higher than those in 2D or 3D (to be shown in Section VI-I).

E. 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 9. A 3D virtual space is used for VPoD. Figure 12 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 12(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 adjustment interval. We used $c_c = 0.1$ for all other experiments presented in this paper.

F. 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$\times$10m square. We varied the number of obstacles from 0 to 10 in the 100m$\times$100m physical space for 200-node networks. The average node degree is reduced. 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 the metric in Figure 13(b). In the same figure, the average number of transmissions of NADV increases by 71.1% from 7.44 (0 obstacle) to 12.73 (10 obstacles), while that of GDV on VPoD (3D) increases by 23.7% 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.

G. Routing performance for sparse networks

We also evaluated GDV on another set of sparse networks, in which the average degree is 8 and two nodes are considered neighbors if the PRR is over 0.33. As shown in Figure 14, GDV in 2D, 3D, and 4D still has better routing performance compared to MDT. Compared to Figure 11, the routing improvement of GDV becomes more significant because greedy routing using physical locations may encounter more local minima and thus perform poorly on sparse networks.

H. Comparison with Vivaldi and PSVC

We compare the routing performance of GDV on VPoD with Vivaldi [6] and PSVC [34]. In Figure 15, we measure the average routing stretch (for the hop count metric) and average number of transmissions per delivery (for the ETX metric) to compare GDV on VPoD, GDV on Vivaldi, and PSVC. Figure 15(a) shows that GDV on VPoD outperforms GDV on Vivaldi by a very large margin. MDT on actual locations and PSVC have similar routing stretch; both of them are close to GDV on VPoD (2D) after convergence. GDV on VPoD (3D) can achieve slightly lower routing stretch than PSVC. However, when ETX is used as the metric (Figure 15(b)), since PSVC has no ability to incorporate link costs, GDV on VPoD is significantly better than PSVC in the average number of transmissions per delivery.

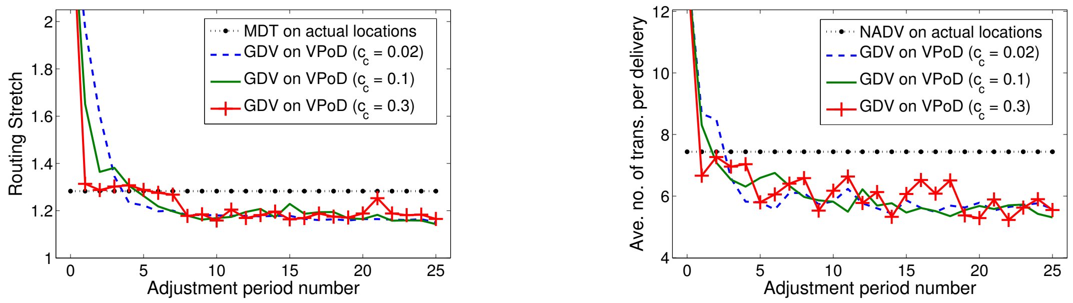


Fig. 12. Routing performance for different values of tuning parameter c_c: **(a)** metric is hop count, **(b)** metric is ETX.

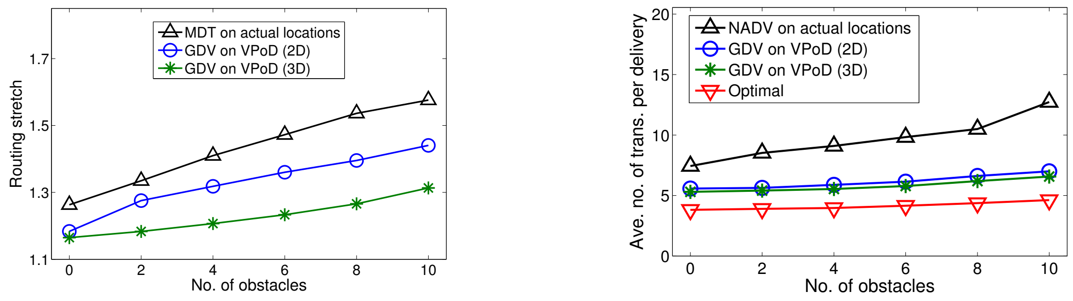


Fig. 13. Routing performance vs. number of obstacles: **(a)** metric is hop count, **(b)** metric is ETX.

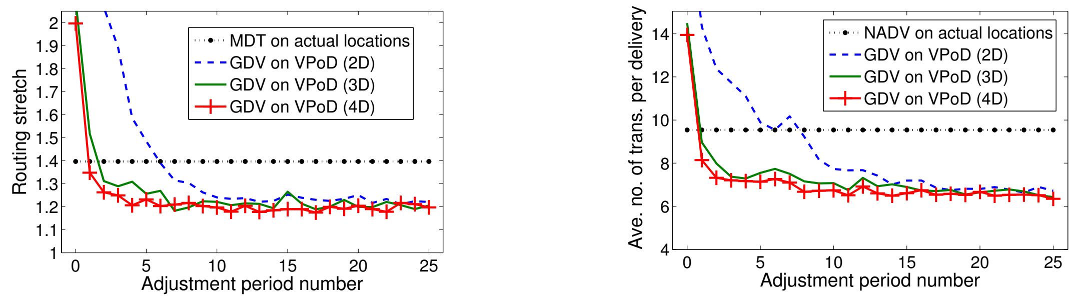


Fig. 14. Routing performance for sparse networks (average degree is 8): **(a)** metric is hop count, **(b)** metric is ETX.

I. 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 9. Figure 16(a) shows storage cost over time. The two 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-greedy and NADV on actual locations. Vivaldi requires much higher storage cost than VPoD. The storage cost of PSVC lies between those of VPoD in 2D and 3D. 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 16(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. After convergence, the message costs of VPoD in 2D and 3D are about 20 and 60 messages, respectively, per join-and-adjustment period. Two-hop Vivaldi requires many more messages. We do not show message costs for MDT-greedy and NADV on actual locations as well as PSVC. Given location information, MDT-greedy and NADV use one-time constructions with low message costs. But they require localization methods which have message and other costs to provide accurate location information. PSVC uses a one-time construction of coordinates and the average number of messages sent per node was 1735, which is higher than the cumulative number of messages used by VPoD at period 15 (543 messages per node for 2D and 1240 messages per node for 3D). Note that VPoD typically

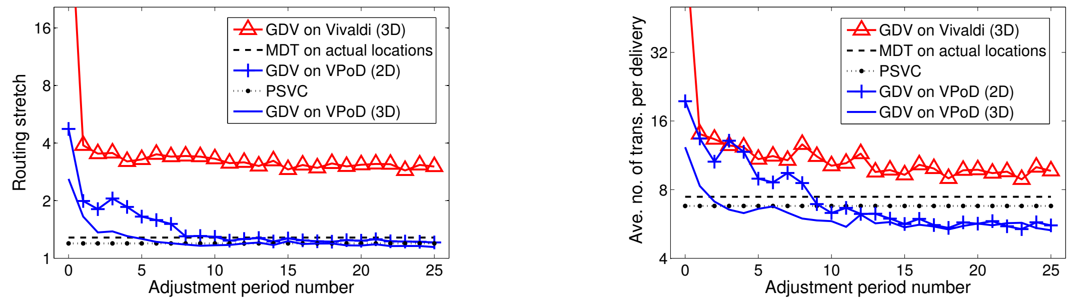


Fig. 15. Routing performance compared with Vivaldi and PSVC: **(a)** metric is hop count, **(b)** metric is ETX.

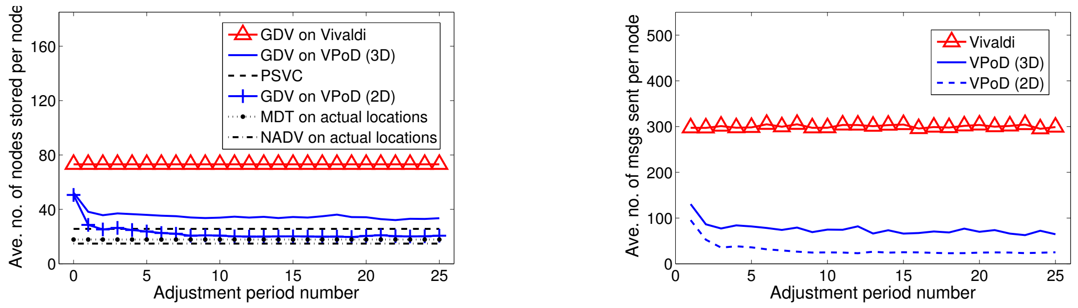


Fig. 16. **(a)** Storage cost and **(b)** control message cost in an adjustment period.

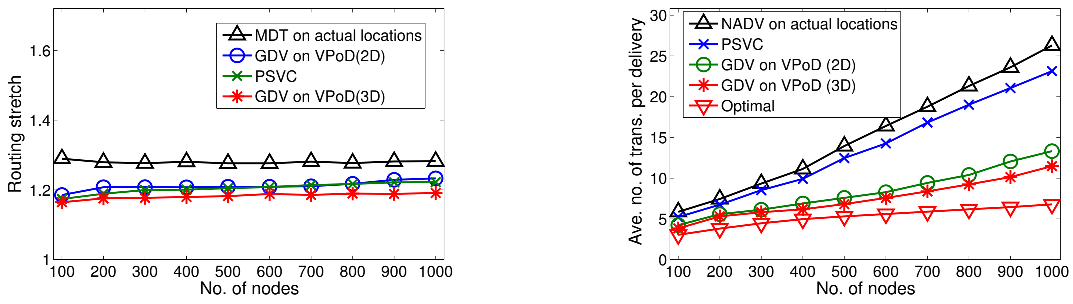


Fig. 17. Routing performance versus N: **(a)** metric is hop count, **(b)** metric is ETX.

converges in 15 adjustment periods.

J. Varying the number of nodes

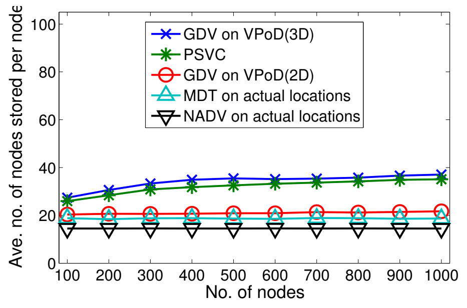
We evaluate the performance of GDV for network size (N) from 100 to 1000 nodes. For 200-node experiments, the size of the physical space is 100m$\times$100m. For a smaller (or larger) number of nodes, the size of the physical space is scaled down (or up) proportionally such that the average number of 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. In these experiments, VPoD stops after 15 adjustment periods.

Figure 17(a) shows routing stretch versus N. GDV on VPoD performs better than MDT-greedy on actual locations. PSVC's routing stretch is close to GDV on VPoD (2D), lower than MDT-greedy on actual locations, and higher than GDV on VPoD (3D). The routing stretch values of GDV, PSVC and MDT-greedy remain low as N increases.

Figure 17(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. Since PSVC cannot incorporate link cost in virtual coordinates, it's routing performance using ETX is similar to NADV and much worse than GDV.

Figure 18(a) shows storage cost versus N. NADV has the lowest cost, followed in order by MDT-greedy, GDV on VPoD (2D), PSVC, and GDV on VPoD (3D). The storage costs for all protocols remain low as N increases.

Figure 18(b) shows the routing success rates of different protocols. GDV, PSVC, and MDT-greedy all 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.



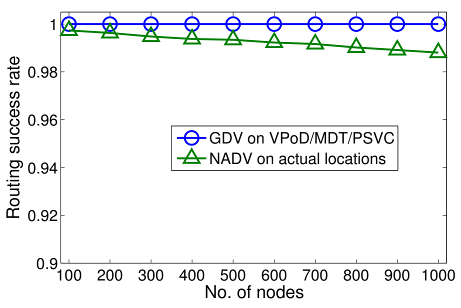
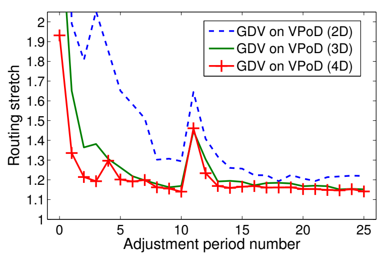
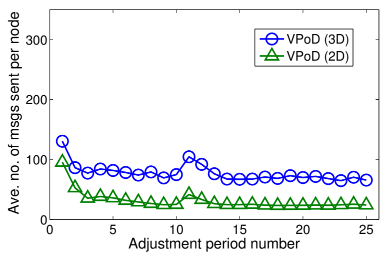


Fig. 18. **(a)** Storage cost and **(b)** routing success rate, versus N





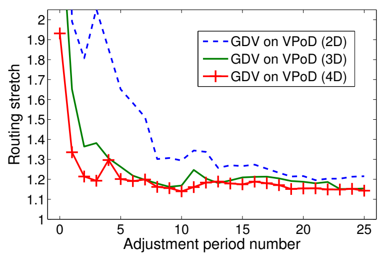


Fig. 19. Routing performance under churn: **(a)** routing stretch, **(b)** control message cost, **(c)** routing stretch under continuous churn.

K. Resilience to dynamic topology changes

GDV and VPoD are highly resilient to dynamic network topology changes (*churn*) because they use MDT protocols which are highly resilient. For the same 200-node network used for the experiments in Figure 11, we introduced node churn 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. Figure 19(a) shows the routing performance of GDV using VPoD in 2D, 3D, and 4D. Note that the routing stretch becomes worse immediately after churn. However, it quickly converges to a low value after several adjustment periods (just 2-3 periods for 3D). The routing stretch after 20 periods in total is as good as the performance shown in Figure 11 for experiments with a static topology. We also show the control message cost to maintain VPoD during the churn in Figure 19(b). Similarly, the message cost increases during churn and also quickly converges to the normal value. In another set of experiments shown in Figure 19(c), churn happens continuously from the 10th period to 20th period. From 10th to 15th period the churn rate is 15 nodes per minute and from 15th to 20th period the churn rate is 30 nodes per minute. Results show that continuous churn has no significant impact on the routing stretch of GDV.

These and similar results from other churn experiments show that GDV and VPoD are very resilient to dynamic topology changes.

VII. Evaluation on Wireline Networks

GDV has minimal assumptions: a connected graph and bidirectional links. Hence GDV can also be applied to wireline networks for intra-domain or layer-2 routing. In this section we evaluate the performance of GDV using real and synthetic wireline network topologies. For wireline experiments, message delivery times from one node to another are sampled from a uniform distribution over the interval $[50\mu s, 150\mu s]$.

We studied three router-level topologies of three ASes collected by the Rocketfuel project [28]. We observed that a real router-level topology usually contains some nodes that have only one or two neighbors, e.g., edge routers. Maintaining a multi-hop DT for a network including these nodes is very inefficient. This is because most DT neighbors of these nodes are multi-hop neighbors, incurring both high communication and storage cost. Furthermore, these nodes do not have enough physical neighbors for position adjustment. To address these problems, we designed two optimization techniques, *pruning* and *two-hop neighbor support*, to improve the performance of GDV and VPoD on network topologies with many low-degree nodes (any node with degree ≤ 3 in our experiments). When a node discovers that it is a low-degree node, it initiates the optimization techniques by sending protocol messages to its physical neighbors.

These optimization techniques can also be used by low-degree nodes in wireless networks. We ran our protocol suite including these optimization techniques on wireless topologies in Section VI. We observed no improvement in GDV and VPoD performance, which is expected, because the number of low-degree nodes in these wireless topologies is zero or insignificant.

Pruning: A node with one physical neighbor does not need to participate in the multi-hop DT, because it has only one link to other nodes. Before starting VPoD, each of these nodes sends a *PRUNED* message to its *parent*, i.e., its only physical neighbor. The *PRUNED* message indicates that the sender

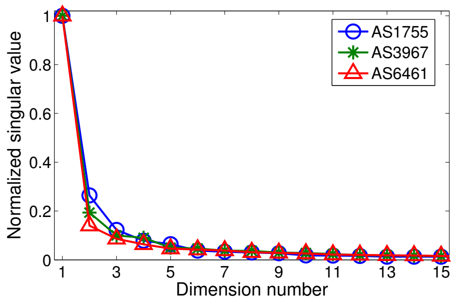


Fig. 20. Normalized singular values for Rocketfuel topologies

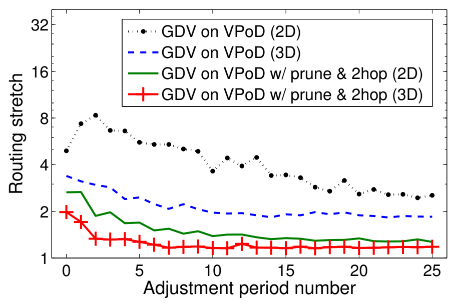
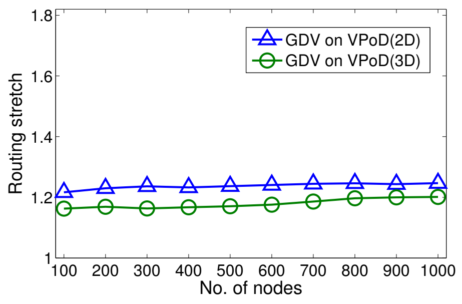


Fig. 21. Routing stretch on AS1755 topology



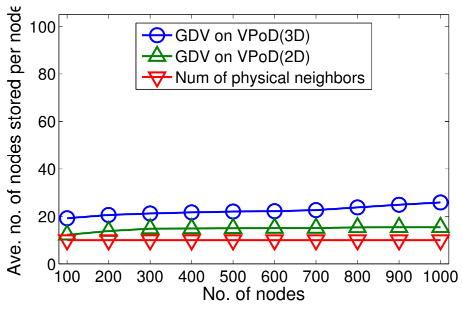


Fig. 22. **(a)** Routing stretch and **(b)** storage cost, versus N (Brite topologies)

declines to participate in the multi-hop DT, and will simply use the parent's position as its own position for receiving data messages. When a node finds that there is only one physical neighbor that is not pruned, it also sends a *PRUNED* message to the physical neighbor. In this way, any tree sub-graph attached to the remaining network topology will be pruned, and only the root participates in the multi-hop DT. Every node in the tree uses the root's virtual position. A data message whose destination is in the tree will be routed to the root and then forwarded. The node-to-node connectivity is guaranteed after pruning because nodes in the tree-like sub-graphs are still reachable. Pruning scales to large networks because it is performed in a local area.

Two-hop neighbor support: For nodes that do not have enough physical neighbors ($\leq$ 3 in the current design) to perform position adjustment, they will select four random two-hop neighbors and include them in their physical neighbor sets. For each two-hop neighbor w stored by node u, the "link cost" $c(u,w)$ is assigned to be $\min_{v \in P_u \bigcap P_w} c(u,v)+c(v,w)$. The $succ$ field in the corresponding tuple is the physical neighbor that minimizes $c(u,v)+c(v,w)$. A two-hop neighbor can be considered the same as a physical neighbor in protocol execution.

We show in Figure 20 the normalized singular values of the routing cost matrices, after pruning, of AS1755, AS3967 and AS6461. The first two or three singular values are much larger than the remaining ones for all three topologies. From the results, VPoD in low dimensions (2D-4D) is good for Rocketfuel topologies.

Figure 21 shows the GDV routing stretch on the AS1755 network in 2D and 3D, with and without pruning and two-hop neighbor support. The original GDV on VPoD has relatively high routing stretch. The converged routing stretch values are about 2.5 and 1.8 for 2D and 3D, respectively. The reason might be that some nodes with few physical neighbors are not able to determine their proper positions. However, pruning and two-hop neighbor support significantly improve the routing stretch. After applying these schemes, the converged values are 1.27 and 1.15 for 2D and 3D respectively. Experiments on AS3967 and AS6461 topologies show similar results.

We then conduct experiments on synthetic router-level topologies generated by the BRITE internet topology generator [19] to evaluate the performance of GDV for networks with different values of N. Each data point shown is the average value of 10 simulation runs for 10 different networks. In these experiments, VPoD stops after 15 adjustment periods. Figure 22(a) shows routing stretch versus N, and Figure 22(b) shows storage cost versus N. Similar to the results of GDV on wireless topologies, the routing stretch and storage cost remain low as N increases, for both 2D and 3D.

VIII. Conclusion

GDV is the *first* greedy 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 which use accurate location information. As a greedy routing 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 both wireline and wireless layer-2 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. GDV and VPoD are highly resilient to network topology changes.

REFERENCES

[1] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly. Symbiotic routing in future data centers. In *Proc. of ACM SIGCOMM*, 2010.

[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] W. Cui and C. Qian. Difs: Distributed flow scheduling for adaptive routing in hierarchical data center networks. In *Proc. of ACM/IEEE ANCS*, 2014.

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

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

[8] 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.

[9] 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.

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

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

[12] 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; extended version in *IEEE/ACM Transactions on Networking*, Vol. 21, No. 2, April 2013.

[13] 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).

[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 et al. Does Wireless Sensor Network Scale? A Measurement Study on GreenOrbs. In *Proceedings of IEEE Infocom*, 2011.

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

[19] A. Medina, A. Lakhina, I. Matta, , and J. Byers. BRITE: An Approach to Universal Topology Generation. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecom Systems*, 2001.

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

[21] C. Qian and S. S. Lam. A Scalable and Resilient Layer-2 Network with Ethernet Compatibility. *Accepted for publication by IEEE /ACM Transactions on Networking*.

[22] C. Qian and S. S. Lam. Greedy distance vector routing. In *Proc. of IEEE ICDCS*, 2011.

[23] C. Qian and S. S. Lam. ROME: Routing On Metropolitan-scale Ethernet. In *Proceedings of IEEE ICNP*, 2012.

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

[25] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. Gu. Greedy routing with guaranteed delivery using ricci flows. In *Proceedings of IPSN*, 2009.

[26] 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.

[27] J.-Y. Shin, B. Wong, and E. G. Sirer. Small-world datacenters. In *Proc. of ACM SOCC*, 2011.

[28] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proceedings of ACM SIGCOMM*, 2002.

[29] 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.

[30] D. Tschopp, S. Diggavi, M. Grossglauser, and J. Widmer. Robust geo-routing on embeddings of dynamic wireless networks. In *Proceedings of IEEE INFOCOM*, 2007.

[31] D. Xu, M. Chiang, and J. Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. *IEEE/ACM Transactions on Networking*, 2011.

[32] Y. Yu and C. Qian. Space shuffle: A scalable, flexible, and high-bandwidth data center network. In *Proc. of IEEE ICNP*, 2014.

[33] 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.

[34] J. Zhou, Y. Chen, B. Leong, and B. Feng. Practical Virtual Coordinates for Large Wireless Sensor Networks. In *Proceedings of IEEE ICNP*, 2010.

Chen Qian (M'08) is an Assistant Professor at the Department of Computer Science, University of Kentucky. He received the B.Sc. degree from Nanjing University in 2006, the M.Phil. degree from the Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from the University of Texas at Austin in 2013, all in Computer Science. His research interests include network protocol design, software defined networking, network functions virtualization, data-center networks, mobile computing, and security. He has published more than 30 research papers in a number of conferences and journals including *ACM SIGMETRICS*, *IEEE ICNP*, *IEEE ICDCS*, *IEEE INFOCOM*, *IEEE PerCom*, *IEEE/ACM Transactions on Networking*, and *IEEE Transactions on Parallel and Distributed Systems*. He is a member of IEEE and ACM.

Simon S. Lam (F'85) received the B.S.E.E. degree with Distinction from Washington State University, Pullman, in 1969, and the M.S. and Ph.D. degrees in engineering from the University of California, Los Angeles, in 1970 and 1974, respectively. From 1971 to 1974, he was a Postgraduate Research Engineer with the ARPA Network Measurement Center at UCLA, where he worked on satellite and radio packet switching networks. From 1974 to 1977, he was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY. Since 1977, he has been on the faculty of the University of Texas at Austin, where he is Professor and Regents Chair in computer science, and served as Department Chair from 1992 to 1994.

He served as Editor-in-Chief of *IEEE/ACM Transactions on Networking* from 1995 to 1999. He served on the editorial boards of *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Software Engineering*, *IEEE Transactions on Communications*, *Proceedings of the IEEE*, *Computer Networks*, and *Performance Evaluation*. He co-founded the ACM SIGCOMM conference in 1983 and the IEEE International Conference on Network Protocols in 1993.

Professor Lam is a Member of the National Academy of Engineering and a Fellow of ACM. He received the 2004 ACM SIGCOMM Award for lifetime contribution to the field of communication networks, the 2004 ACM Software System Award for inventing secure sockets and prototyping the first secure sockets layer (named Secure Network Programming), the 2004 W. Wallace McDowell Award from the IEEE Computer Society, as well as the 1975 Leonard G. Abraham Prize and the 2001 William R. Bennett Prize from the IEEE Communications Society.