Beacon Vector Routing: Scalable Point-to-Point Routing in Wireless Sensornets

Rodrigo Fonseca* Sylvia Ratnasamy† Jerry Zhao‡ Cheng Tien Ee*
David Culler* Scott Shenker*,‡ Ion Stoica*

*University of California, Berkeley {rfonseca, ct-ee, culler, istoica}@cs.berkeley.edu
†Intel Research, Berkeley sylvia@intel-research.net
‡International Computer Science Institute {zhao, shenker}@icsi.berkeley.edu

Abstract

We propose a practical and scalable technique for point-to-point routing in wireless sensornets. This method, called Beacon Vector Routing *(BVR), assigns coordinates to nodes based on the vector of hop count distances to a small set of beacons, and then defines a distance metric on these coordinates. BVR routes packets greedily, forwarding to the next hop that is the closest (according to this beacon vector distance metric) to the destination. We evaluate this approach through a combination of high-level simulation to investigate scaling and design tradeoffs, and a prototype implementation over real testbeds as a necessary reality check.*

1 Introduction

The first generation of sensornet deployments focused primarily on data collection [23, 9]. In support of this task, most current sensornet code bases [11, 7] offer only the basic tree-based many-to-one and one-to-many routing primitives; protocols such as Directed Diffusion [12], TAG[21], and others build trees that can both broadcast commands and collect data, with various forms of aggregation along the collection path. However, a growing number of recent proposed uses require more sophisticated point-to-point routing support. These include applications such as PEG (a pursuer-evader game in which a large network tracks the movement of evader robots [4]), approaches such as reactive tasking (commands based on local sensing results), and data query methods such as multi-dimensional range queries [20], spatial range queries[8], and multi-resolution queries [6], and data-centric storage [29].

Unfortunately, it is hard to test these ideas because there is currently no practical and broadly-applicable implementation of point-to-point routing for sensornets. We know of two implementations of a reduced AODV and of GPSR [15], but they haven't been reported on in the literature. As we discuss in the next section, they have limitations on their applicability. It isn't clear how important these newly proposed uses are, but without a point-to-point routing protocol we will never be able to evaluate their true utility. Moreover, the applications and services that emerge from the sensornet community will depend, in part, on which routing primitives have scalable and practical implementations; hence, the lack of a robust implementation of point-to-point routing might well limit the scope of future sensornet applications.

The lack of point-to-point implementations is in stark contrast with the bevy of proposed designs in this space. As we review in the next section, there have been many different approaches to this problem, but none has resulted in a reliable implementation. We speculate that this is largely due to the clash between the complexity of these proposals and the demanding requirements of sensornet implementation. Sensornet implementations should not only meet stringent scaling, robustness, and energy efficiency standards, but they should also function on a hardware base that has severe resource limitations (in terms of memory and packet length) and varying quality radios. The impact of these factors on design is best illustrated by the experience of the TinyOS developers (described in [19]) where the algorithmically trivial flooding and tree construction primitives took three years and five successive implementations to get right.

Thus, simplicity is our primary design requirement. We make minimal assumptions about radio quality, presence of GPS, and other factors, and want minimal complexity in the algorithm itself. We do so by using the previous hard-won successes in tree-building as the basic building block of our more general routing protocol. We select a few *beacon* nodes and construct trees from them to every other node (using standard techniques). As a result, every node is aware of its distance (in hops) to every beacon and these beacon vectors can serve as coordinates. After defining a distance metric over these coordinates, we can use a simple greedy distance-minimizing routing algorithm. This approach, which we call Beacon Vector Routing (BVR), requires very little state, overhead, or pre-configured information (such as geographic location of nodes). Routes are based on connectivity, which nodes are naturally aware of, and in our simula-

tions and measurements appear to be reasonably close to the minimal distance paths.

The remainder of this paper is organized as follows: we compare BVR to related work in Section 2, trying to illustrate both qualitative and quantitative differences between the various proposals. We present the BVR routing algorithm in Section 3 and use high-level simulations to investigate scaling and design tradeoffs in Section 4. We describe the details of the BVR implementation in Section 5 and evaluate our implementation with two independent testbeds (to provide a much-needed reality check on our results) in Section 6. Future directions and our conclusions are presented in section 7.

2 Related Work

The many proposals for point-to-point routing in ad-hoc wireless networks [26] can be broadly divided into four very different categories. We discuss each of these in turn, highlighting their pros and cons when applied to sensor networks, and then contrast them with BVR. Table 1 summarizes our discussion.

Shortest Path: This is the classical approach to routing in which a distributed form of Dijkstra's algorithm is used to compute the shortest path between a source and destination. In early protocols such as Distance-Vector, Link-State and DSDV, the shortest path between all possible source-destination pairs is computed and every node stores its next hop to every destination. For a network with n nodes, this results in O(n^2) message exchanges for route discovery and O(n) routing state at each node. This overhead, particularly the per-node state, scales poorly to large networks. For example, a Mica2 mote has only 4KB RAM; in a 1000 node network, a node's routing table alone would exhaust this.

To reduce this overhead, Johnson *et al.* proposed the use of *on-demand* route discovery [13]. The resulting improvement in scalability depends entirely on the overall traffic pattern and, while these protocols perform admirably in many settings, they are not well-suited to cases with traffic between many source-destination pairs (which can be expected in DIM [20], PEG [4], Dimensions [6], *etc.*).

Hierarchical Addressing: The (wired) Internet uses careful address allocation which allows significant route aggregation and thus smaller routing tables. This is infeasible in sensor networks in part because of the overhead of manual configuration but also because a sensornet's connectivity graph is dependent on the details of its physical environment and is often quite variable; this makes it difficult to determine *a priori* how addresses should be assigned.

Francis' [31] elegant Landmark Routing (LR) proposal solves this problem by allowing nodes to self–configure their addresses. LR uses a hierarchical set of landmark nodes that periodically send scoped route discovery messages. A node's address is the concatenation of its closest landmark at each level in the hierarchy. LR reduces the overhead of route setup to O($n \log n$) and nodes only hold state for their immediate neighbors and their next hop to each landmark. However, LR requires a protocol that creates and maintains this hierarchy of landmarks and appropriately tunes the landmark scopes. The original LR proposal does not address the details of such a protocol and no workable implementation has been deployed. More recent proposals adopting this approach have been fairly complex [17], in conflict with our design goal of configuration simplicity.

Geographic Coordinates: A different and potentially attractive solution for sensor networks is based on geographic routing [15, 1, 16]. Here, nodes are identified by their geographic coordinates and routing is done greedily; at each step, nodes pick as next-hop the neighbor that is closest to the destination. When a node has no neighbor that is closer to the destination, these protocols enter perimeter mode, where the right-hand rule is used to forward a packet along a planarized subgraph until it reaches a node closer to the destination than the starting point of perimeter mode (then it resumes its greedy forwarding). Geographic routing is eminently scalable— it incurs $O(1)$ overhead for route discovery and $O(1)$ routing tables (a node need only discover and store its one-hop neighbors), the planarization techniques are purely local, and path lengths are close to the shortest path [15]. Unfortunately, geographic routing has two problems. First, the correctness of the common (local) planarization algorithms, and hence the correctness of perimeter mode routing, relies on a unit-graph assumption under which a node hears all transmissions from nodes within its fixed radio range and never hears transmissions from nodes outside this range. Measurement studies [32, 33, 3] have shown that this assumption is grossly violated by real radios. Second, and more seriously, such routing requires that each node know its geographic coordinates. While there are some sensor nodes that are equipped with GPS, the most widely used node, the Berkeley mote [10], is not. Moreover, even when available, GPS does not work in certain physical environments and the various proposed localization algorithms [27] are not precise enough (at least not in all settings) to be used for geographic routing. Finally, even ignoring all the above, greedy geographic may be substantially suboptimal because it does not use real connectivity information and geography is, as we show in Section 6, not always in congruence with true network connectivity (*e.g.*, in the face of obstacles or consistent interference).

Virtual Coordinates: Motivated by the ideal scaling properties of schemes like GPSR, two recent proposals

Metric	DV/LS	AODV/DSR	Landmark	GPSR	NoGeo	GEM	BVR
Setup overhead	O(n^2)	n/a	O($nlogn$)	—	O($n\sqrt{n}$)	O(dn)	O(rn)
Route overhead	1	O(n) if route is uncached; else O(1)	$>$ O(1)	O(1)	O(1)	O(1)	O(1)
Per-node State	O(n)	depends on traffic pattern	O($d + logn$)	O(d)	O(d)	O(d)	O($d + r$)
Delivery Guar.	yes	yes	yes	assuming unit graph topologies	yes	yes	yes
Configuration	no	no	hierarchically scoped landmarks	geographic positions	perimeter discovery	coordinate construction	pick r random nodes
Local recovery	? (limited)	no	? (limited)	yes	yes	yes	yes
Local repair	no	no	no	yes	mostly	no	no

Table 1: **Design considerations for sensornet routing algorithms and the tradeoffs using current solutions. This table is intended to be illustrative rather than definitive.** *n is the number of nodes. Setup overhead refers to the total message traffic generated to setup pairwise routes. While to some extent a one-time cost, this is also indicative of the overhead incurred by topology changes. Route overhead refers to the number of transmissions relative to the optimal shortest path. Per-node state is the number of routing entries maintained at each node. Delivery guarantee indicates whether a solution guarantees, at the* algorithmic *level, whether the protocol is guaranteed to find a route to all destinations. Local recovery refers to a node's ability to route around failed nodes in the absence of* any *recovery protocol. Local repair refers to a protocol's ability to limit the impact of node failure to that node's immediate neighbors. d denotes a node's degree (immediate neighbors) and r denotes the number of beacons in BVR, typically a small constant (10). A question mark indicates that our uncertainty about the claimed performance as the literature may not have addressed the relevant consideration.*

attempt to use geographic routing ideas without requiring geographic coordinates. The NoGeo scheme [25] creates synthetic coordinates through an iterative relaxation algorithm that embeds nodes in a Cartesian space. The initialization technique for this scheme requires roughly $O(\sqrt{n})$ nodes to flood the network, and for each of these flooding nodes to store the entire $O(\sqrt{n} \times \sqrt{n})$ matrix of distances (in hops). This is keeping $O(n)$ state at roughly $O(\sqrt{n})$ nodes, an impractical burden in large networks. GEM [24] uses a more scalable initialization scheme but employs an intricate recovery process in which, when nodes fail or radio links degrade, a potentially large number of nodes in the system must recompute routing labels so as to maintain GEM's regular topological structure. Neither NoGeo nor GEM have been implemented on any hardware platform and, while they represent significant conceptual advances, the complexity of coordinate construction and maintenance in these schemes is likely to render both quite difficult to implement and operate in practice.

BVR: BVR borrows, and differs, from each of the above. BVR incurs smaller routing state than the shortest path algorithms (constant vs. O(n)). From Landmark Routing, BVR borrows from the notion of using landmarks to infer node addresses, though the details of the addressing and forwarding are entirely different. Moreover, BVR's beacons are randomly chosen and need not adhere to any particular structure. BVR uses greedy forwarding over node coordinates, but (unlike GPSR) does not require geographic information, makes no assumptions about radio connectivity, and (unlike NoGeo and GEM) uses a very simple coordinate construction algorithm. As mentioned earlier, the core mechanism in BVR is the construction of reverse path trees similar to [22, 34]. However, unlike these schemes, BVR does not directly route along these trees and instead supports point-to-point communication.

We have recently become aware of Logical Coordinate Routing [2], developed simultaneously and independently of BVR, which employs the same idea of nodes obtaining coordinates from a set of landmarks and routing to minimize a distance function on these coordinates. The main difference is the alternative method of routing when local minima are reached in greedy routing: they backtrack the packet along the path, until a suitable path is found, or the route fails at the origin. While they never have to resort to small scoped floods, as does BVR, their algorithm does require that the nodes keep a record of all packets forwarded recently, increasing the amount of state in the nodes.

3 The BVR Algorithm

BVR defines a set of coordinates and a distance function to enable scalable greedy forwarding. These coordinates are defined in reference to a set of "beacons" which are a small set of randomly chosen nodes; using a fairly standard reverse path tree construction algorithm every node learns its distance, in hops, to each of the beacons. A node's coordinates is a vector of these distances. On the occasion that greedy routing with these coordinates fails, we use a correction mechanism that guarantees delivery.

Let q_i denote the distance in hops from node q to beacon i. Let r denote the total number of beacon nodes. We define a node q's position $\mathcal{P}(q)$ as being the vector of these beacon distances: $\mathcal{P}(q) = \langle q_1, q_2, \cdots, q_r \rangle$. Two nodes can have the same coordinates, so we always retain a node identifier to disambiguate nodes in such cases. Nodes must know the positions of their neighbors

Packet fields	Description
$pkt.dst$	the destination's unique identifier
$pkt.\mathcal{P}(dst)$	destination's BVR position
$pkt.\overline{\delta^{min}}$	δ_i^{min} seen, $i \in 1, ..., k$

Table 2: BVR packet header fields

to make routing decisions, so nodes periodically send a local broadcast messages announcing their coordinates.

To route, we need a distance function $\delta(p, d)$ on these vectors that measures how good p would be as a next hop to reach a destination d. The goal is to pick a function so that using it to route greedily usually results in successful packet delivery. The metric should favor neighbors whose coordinates are more similar to the destination. Minimizing the absolute difference component-wise is the simplest such metric. The key piece of intuition driving our design is that it is more important to move *towards* beacons than to move *away* from beacons. When trying to match the destination's coordinates, we move towards a beacon when the destination is closer to the beacon than the current node; we move away from a beacon when the destination is further from the beacon than the current node. Moving towards beacons is always moving in the right direction, while moving away from a beacon might be going in the wrong direction (in that the destination might be on the other side of the beacon). To embody this intuition, we use the following two sums:

$$\delta_k^+(p, d) = \sum_{i \in C_k(d)} \max(p_i - d_i, 0) \quad \text{and}$$

$$\delta_k^-(p, d) = \sum_{i \in C_k(d)} \max(d_i - p_i, 0),$$

where $C_k(d)$ is the set of the k closest beacons to d. δ_k^+ is the sum of the differences for the beacons that are closer to the destination d than to the current routing node p, while δ_k^- measures the sum of the distances to the farther beacons. We choose the next hop that minimizes δ_k^+ and, when there is a tie, we break it by minimizing δ_k^-. In practice, we implement this by minimizing the sum $\delta_k = A\delta_k^+ + \delta_k^-$ for some sufficiently large constant A. In our implementation we use $A = 10$. In addition, δ_k only considers the k closest beacons to d. This serves to reduce the number of distance elements d_i that must be carried in the packet, and is consistent with the idea of moving towards close beacons.

To route to a destination dst, a packet has three header fields, summarized in Table 2: (1) the destination's unique identifier, (2) it's position $\mathcal{P}(dst)$ defined over the beacons in $C_k(dst)$, and (3) $\overline{\delta_{min}}$, a k-position vector where δ_i^{min} is the minimum δ that the packet has seen so far using $C_i(dst)$, the i closest beacons to dst. δ_i^{min} can guarantee that the route will never loop.

Algorithm 1 lists the pseudo-code for BVR forwarding. The parameters are r, the total number of beacons, and $k \leq r$, the number of beacons that define a destination's position. Forwarding a message starts with a greedy search for a neighbor that improves the minimum distance we have seen so far. When forwarding the message, the current node (denoted $curr$) chooses among its neighbors the node $next$ that minimizes the distance to the destination. We start using the k closest beacons to the destination, and if there is no improvement, we successively drop beacons from the calculation.

Algorithm 1 BVR forwarding algorithm

```
BVR_FORWARD(node curr, packet P)
  // first update packet header
  for (i = 1 to k) do
    P.δ_i^min = min(P.δ_i^min, δ_i(curr, P.dst))

  // try greedy forwarding first
  for (i = k to 1) do
    next ← argmin_{x∈NBR(curr)}{δ_i(x, P.dst)}
    if (δ_i(next, P.dst) < P.δ_i^min ) then
      unicast P to next

  //greedy failed, use fallback mode
  fallback_bcn ← closest beacon to P.dst
  if (fallback_bcn != curr) then
    unicast P to PARENT(fallback_bcn)

  //fallback failed, do scoped flood
  broadcast P with scope P.P(dst)[fallback_bcn]
```

In some situations greedy forward may fail, in that no neighbor will improve on δ_i^{min} for any i. We use a 'fallback' mode to correct this. The intuition behind fallback mode is that if a node cannot make progress towards the destination itself, it can instead forward towards a node that it knows is close to the destination and towards which it does know how to make progress. The node forwards the packet towards the beacon closest to the destination; *i.e.*, to its parent in the corresponding beacon tree. The parent will forward as usual – first trying to forward greedily and, failing to do so, using fallback mode.

A packet may ultimately reach the beacon closest to the destination and still not be able to make greedy progress. At this point, the root beacon initiates a scoped flood to find the destination. Notice that the required scope of the flood can be precisely determined – the distance in hops from the flooding beacon to the destination is determined from the destination's position in the packet header. While this ensures that packets can always reach their destination, flooding is an inherently expensive operation and hence we want to minimize the frequency with which it is performed, and also its scope. Our results show both these numbers to be low.

Beacon Maintenance Sensor network nodes are prone to failure and we must provide a mechanism to maintain

the set of beacons when they fail. We first note that the algorithm we described can function with fewer than r beacons, and even when there is inconsistency in the beacon sets nodes are aware of, by routing only based on the beacons they have in common. Thus, the beacon maintenance need not be perfect, it only needs to guide the system towards a state where there are r globally recognized beacons. We now sketch such an algorithm. For convenience, we describe the simplest algorithm we've used; we've also experimented with more advanced algorithms but describing them would take us too far afield.

To detect beacon failures, each entry in the beacon vector is associated with a sequence number. Beacons periodically advance their sequence number; if a node detects that a sequence number has not been updated with a given timeout period then it deletes that beacon from the set it uses to route (note that this decision need not be globally consistent). When the number of beacons alive falls below a configurable parameter r, non-beacon nodes will nominate themselves as beacons. Using ideas from SRM [5], each node sets a timer that is a function of its unique identifier and, when the timer expires, it starts acting like a beacon. If it detects that there are more than r beacons with identifiers smaller than its identifier, it then ceases to be a beacon. Algorithm 2 shows our beacon selection algorithm. More sophisticated beacon maintenance protocols can be designed to more fully optimize beacon placement and suppression.

Algorithm 2 BVR beacon maintenance algorithm

BEACON_ELECT_MYSELF(r, B)
  // invoked periodically; B is the current set of beacons
  if ($|B| > r$) **then**
    set timer $T = \frac{log(myID)}{\log(maxID(B))} * T_{max} + jitter$

TIMER_T_EXPIRES(r, B)
  if ($|B| < r$) **then**
    Announce myself as a beacon

BEACON_SUPPRESS_MYSELF(r, B)
  if ($myID \in B$)&($|B| > r$)&($myID > r^{th}$_guested(B)) **then**
    Stop announcing myself as a beacon

Location Directory Our description so far assumes that the originating node knows the coordinates of the intended destination. Depending on the application, it may be necessary for the originating node to first look up the coordinates by name. We describe a simple mechanism to map node identities to its current coordinates, although this is not the focus of this paper. We propose to use the beacons as a set of storage nodes, by using consistent hashing [14] to provide a mapping $\mathcal{H} : nodeid \mapsto beaconid$, from node ids to the set of beacons. As all nodes know all beacons, any node can independently (and consistently) compute this mapping. The location service consists of two steps: each node k that wishes to be a destination periodically publishes its coordinates to its corresponding beacon $b_k = \mathcal{H}(k)$. Publishing the information entails a self-lookup, which serves as a confirmation. If the coordinates do not change, nodes may choose to refresh their coordinates at a very low rate. Even with changes, we rate limit the update traffic. When a node i wants to route to k, it sends a lookup request to the beacon b_k. Upon receiving a reply, it then routes to the received coordinates. Further communication between the nodes may skip the lookup phase by caching or piggybacking their own location information on the packets they send. We expect that typical data exchanges will be significantly greater in size than these lookup exchanges, so we don't expect that lookup traffic will be a dominant source of sensornet traffic.

This soft-state based approach allows two mechanisms to recover from beacon failures. First, the hashing scheme allows the deterministic choice of backup beacons to replicate the information. The degree of replication depends on the expected failure rate of beacons. Second, the periodic updates will naturally populate a newly elected beacon that replaces a failed beacon. It is important to note that given the redundancy of the coordinate system, even slightly outdated information will lead the routes close to the destination. As we show in our experimental results, both the magnitude and the frequency of the changes to coordinates is small in practice.

4 Simulation Results

To evaluate the BVR algorithm, we use extensive simulations and experiments on testbeds of real sensor motes. To aid the development of BVR and to better understand its behavior and design tradeoffs we start by evaluating BVR using a high-level simulator that abstracts away many of the vagaries of the underlying wireless medium. While clearly not representative of real radios, these simplifications allow us to explore questions of algorithm behavior over a wide range of network sizes, densities, and obstacles that would not be possible on a real testbed.

In practice however, the characteristics of wireless sensor networks impose a number of challenges on actual system development. For example, the `mica2dot` motes have severe resource constraints – just 4KB of RAM, typical packet payloads of 29 bytes *etc.* – and the wireless medium exhibits changing and imperfect connectivity. Hence, our next round of evaluation is at the actual implementation level. We present the implementation and experimental evaluation of our BVR prototype in Sections 5 and 6 respectively and our simulation results in this section.

Our simulator makes several simplifying assumptions. First, it models nodes as having a fixed circular radio range; a node can communicate with all and only those nodes that fall within its range. Second, the simulator

ignores the capacity of, and congestion in, the network. Finally, the simulator ignores packet losses. While these assumptions are clearly unrealistic, they allow the simulator to scale to tens of thousands of nodes. We place nodes uniformly at random in a square planar region, and we vary the total number of beacons r, and the number of routing beacons, k. In all our tests, we compare the results of routing over BVR coordinates to greedy geographic routing over the true positions of the nodes.

Our default simulation scenario uses a 3200 node network with nodes uniformly distributed in an area of 200 $\times$ 200 square units. The radio range is 8 units, and average node degree is 16. Unless otherwise stated, a node's neighbors are those nodes within its one hop radius.

4.1 Metrics

In our evaluation, we consider the following performance metrics:

(Greedy) success rate: The fraction of packets that are delivered to the destination **without requiring flooding**. We stress that the final scoped flooding phase ensures that **all** packets eventually reach their destination. This metric merely measures how often the scoped flooding is not required. Like previous virtual coordinate solutions [24, 25], we report on the success of routing without scoped floods because that provides the most unambiguous evaluation of the quality of node coordinates themselves; *e.g.*, scoped flooding, which will always succeed, does not depend on coordinates. If our results are comparable to those with true positions, then BVR would have overcome the need for geographic information in current proposals. Nonetheless, later in this section, we also present results on a metric we term *transmission stretch* that does explicitly account for the overhead of scoped floods.

Flood scope: The number of hops it takes to reach the destination in those cases when flooding is invoked.

Path stretch: The ratio of the path length of BVR to the path length of greedy routing using true positions.

Node load: The number of packets forwarded per node.

In each test, we're interested in understanding the overhead required to achieve good performance as measured by the above metrics. There are three main forms of overhead in BVR:

Control overhead: This is the total number of flooding messages generated to compute and maintain node coordinates and is directly dependent on r, the total number of beacons in the system. We measure control overhead in terms of the total number of beacons that flood the network. Ideally, we want to achieve high performance with reasonably low r.

Per-packet header overhead: A destination is defined in terms of its $k(\leq r)$ routing beacons. Because the destination position is carried in the header of every packet for routing purposes, k should be reasonably low.

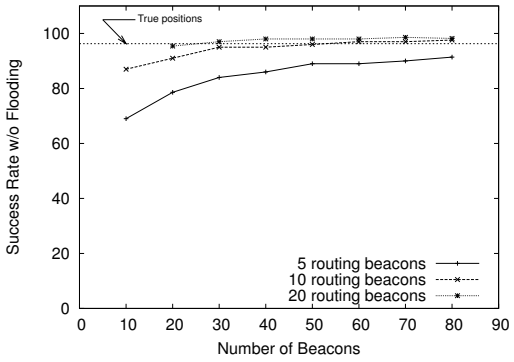

Figure 1: Success rate of routes without flooding in a 3200 node network, for different numbers of total beacons, r, and routing beacons, k.

Routing state: The number of neighbors a node maintains in its routing table.

4.2 Routing Performance vs. Overhead

In this section, we consider the tradeoff between the routing success rate and the flood scope on one hand, and the overhead due to control traffic (r) and per-packet state (k) on the other hand. We use our default simulation scenario and for each of ten repeated experiments, we randomly choose r beacons from the total set of nodes. We vary r from 10 to 80 each time generating $32,000$ routes between randomly selected pairs of nodes.

Figure 1 plots the routing success rate for an increasing total number of beacons (r) at three different values of k, the number of routing beacons ($k = 5$, 10, and 20) As expected, the success rate increases with both the number of total beacons and the number of routing beacons. We draw a number of conclusions from these results. We see that with just $k = 10$ routing beacons we can achieve routing performance comparable to that using true positions. The performance improvement in increasing k to 20 is marginal. Hence, from here on, we limit our tests to using $k = 10$ routing beacons as a good compromise between per-packet overhead and performance. Using $k = 10$, we see that only between 20 to 30 total beacons (r) is sufficient to match the performance of true positions. At less than 1% of the total number of nodes, this is very reasonable flooding overhead. The scope of floods as a function of r decreases from 7 at r =10 to 3 at r =70.

The average path length in these tests was 17.5 hops and the path stretch, i.e., the length of the BVR path over the path length using greedy geographic routing over true positions, is 1.05. In all our tests, we found that the path stretch was always less than 1.1 and hence we don't present path stretch results from here on.

We also compared the distribution of the routing load over nodes using BVR versus greedy geographic routing over true positions and found that for most nodes,

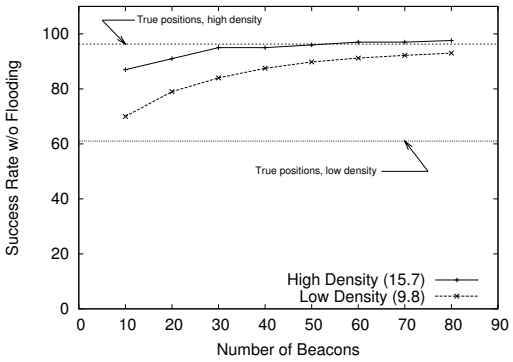

Figure 2: Success rate of routes without flooding, for 3200 node topologies with different densities, for $k = 10$ routing beacons.

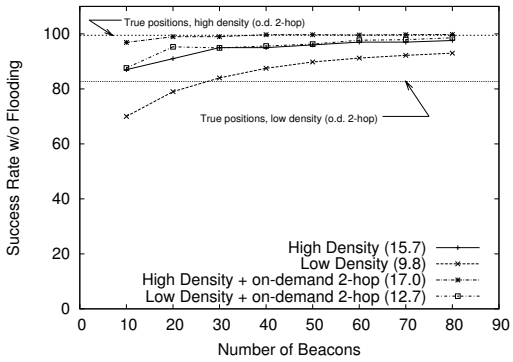

Figure 3: Success rate of routes without flooding, for the same topologies as in figure 2, comparing the on demand acquisition of 2-hop neighborhood information.

the load is virtually identical though BVR does impose slightly higher load on the nodes in the immediate vicinity of beacons. For example, for the above test using $r = 40$ and $k = 10$, the 90th percentile load per node was 48 messages using BVR compared to 37 messages using true positions.

In summary, we see that BVR can roughly match the performance of greedy geographic routing over true positions with a small number of beacons using only its one-hop neighbors.

4.3 The Impact of Node Density

In this section, we consider the impact of the node density on the routing success rate. Figure 2 plots the success rate for the original density of 16 nodes per communication range, and for a lower density of 9.8 nodes per communication range. While at high density the performance of both approaches is comparable, we see that at low densities BVR performs much better than greedy geographic routing with true positions. In particular, while the success rate of the greedy routing is about 61%, the success rate of BVR reaches 80% with 30 beacons, and 90% with 40 beacons. Thus, BVR achieves an almost 30% improvement in the success rate compared to greedy routing with true positions. This is because the node coordinates in BVR are derived from the connectivity information, and not from their geographic positions which may be misleading in the presence of the voids that occur at low densities.

These results reflect the inherent tradeoff between the amount of routing state per node and the success rate of greedy routing. At lower densities, each node has fewer immediate neighbors and hence the performance of greedy routing drops. One possibility to improve the performance of our greedy routing is to have nodes maintain state for nodes beyond their one-hop neighborhood. This however increases the overhead and complexity of maintaining routing state. To retain high success rates without greatly (or needlessly) increasing the routing state per node, we propose the use of *on-demand* two-hop neighbor acquisition. Under this approach, a node starts out using only its immediate (one-hop) neighbors. If it cannot forward a message greedily, it fetches its immediate neighbors' neighbors and adds this two-hop neighbors to its routing table. The intuition behind this approach is that the number of local minima in a graph is far smaller than the total number of nodes. Thus, the on-demand approach to augmenting neighbor state allows only those nodes that require the additional state to incur the overhead of maintaining this state.

Algorithm	avg ngbrs	max ngbrs	% nodes w/ 2hop	avg success
BVR (hi-dens)	15.7	30.7	0	96.1
BVR+2hop (hi-dens)	17.0	67.5	5	99.7
true postns (hi-dens)	15.7	31.7	0	96.3
true postns+2hop (hi)	15.8	48.0	0.7	99.5
BVR (lo-dens)	9.8	22.1	0	89.2
BVR+2hop (lo-dens)	12.7	50.0	15	97.0
true postns (lo-dens)	9.8	22.8	0	61.0
true postns+2hop (lo)	10.7	36.3	6	82.7

Table 3: State requirements using on-demand two hop neighbor acquisition for BVR and true positions at two different network densities. These state requirements are averaged over 10 runs with $k = 10$ and $r = 50$.

To evaluate the effectiveness of using on-demand two-hop neighbor acquisition, we repeat the experiments in Figure 2 using this approach. The results are plotted in Figure 3. Not surprisingly, this approach greatly improves the routing success rate. With only 20 beacons, the success rate of BVR exceeds 99% for the high density network, and 96% for the low density network. Table 3 shows the average and worst case increase in the per-node routing state for both BVR and true positions. Using BVR, at high density, only 5% of nodes fetch their two-hop neighbors while 15% of nodes do so at the lower densities. Thus acquiring two-hop neighbors on demand represents a big win at a fairly low cost.

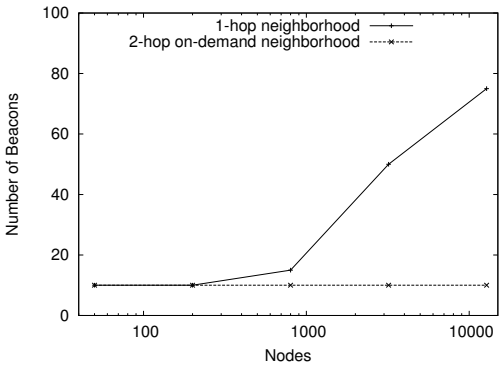

Figure 4: Number of beacons required to achieve less than 5% of scoped floods, with $k = 10$ routing beacons.

4.4 Scaling the Network Size

In this section, we ask the following question: how many beacons are needed to achieve a target success rate as the network size increases? To answer this question, we set the target of the routing success rate at 95%. Figure 4 plots the number of beacons required to achieve this target for both BVR using a one-hop neighborhood, and BVR using on-demand two-hop neighbor acquisition. In both cases the number of routing beacons is 10.

There are two points worth noting. First, the number of beacons for the on-demand two-hop neighborhood remains constant at 10 as the network size increases from 50 to $12,800$ nodes. Second, while the number of beacons in the case of BVR with one-hop neighborhood increases as the network size increases, this number is still very small. When the network is greater than 800 nodes, the number of beacons for the one-hop neighborhood never exceeds 2%.

These results show that the number of beacons required to achieve low flooding rates grows slowly with the size of the system.

4.5 Performance under obstacles

We now study the BVR performance in the presence of obstacles. We model obstacles as horizontal or vertical "walls" with lengths of 10 or 20 units. For comparison, recall that the radio range of a node is 8 units.

Table 4 shows the success rates of BVR routing over a one-hop neighborhood for different numbers of obstacles. For each entry, we also show, in parentheses, the success rate of greedy routing using true positions. Surprisingly, as the number of obstacles and/or their length increases, the decrease in success rate using BVR is *not* significant. In the worst case the success rate drops only from 96% to 91%. For comparison, the success rate of greedy routing with true positions drops from 98% to 43%! Again, this is because the node coordinates in BVR reflect their connectivity instead of their true positions.

Length of Obstacles	Number of Obstacles			
	0	10	20	50
10	0.96 (0.98)	0.96 (0.91)	0.95 (0.87)	0.95 (0.79)
20	0.96 (0.98)	0.95 (0.84)	0.94 (0.70)	0.91 (0.43)

Table 4: Comparing BVR with greedy forwarding over true positions in the presence of obstacles

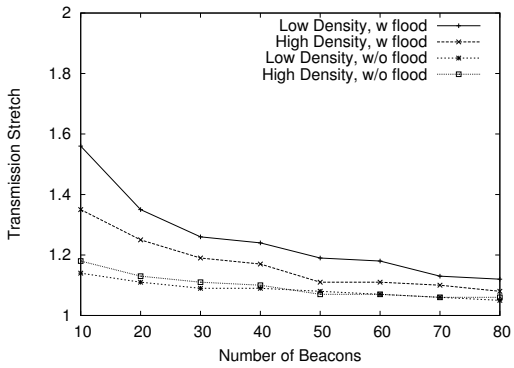

Figure 5: Transmission stretch, average total number of messages sent per route over the the number of messages sent over the shortest path.

4.6 Transmission Stretch

Our results so far evaluated the success of routing without scoped floods. Because scoped flooding incurs higher messaging overhead than unicast forwarding, we now look at a metric we call *transmission stretch*. We measure transmission stretch as the ratio of the *total* (unicast and scoped-flood) number of messages transmitted in routing a packet to that required using the optimal shortest path as computed by Dijkstra's algorithm. Figure 4.5 plots this stretch for BVR routing with and without the use of scoped flood. In the absence of scoped floods we compute stretch only over those routes that do not require floods. We can see that at both low and high density, the transmission stretch improves with increasing number of beacons and rapidly drops to very close to 1. This shows that the use of scoped floods does not incur significant additional overheads. We repeated these tests for network sizes from 50 to 3200 and found that in all cases, the stretch was less than 1.1.

5 BVR Implementation

This section describes our prototype implementation of BVR in TinyOS [11] for the `mica2dot` motes. The resource constraints of the mote hardware and the vagaries of the wireless medium lead to a number of practical difficulties not addressed in our discussion so far. In particular, the following are four key issues that must be addressed in a real implementation:

Link estimation: In a wireless medium, the notion of an individual link is itself ill-defined as the quality of communication varies dramatically across nodes, distance and time. Link estimation is used to characterize a link

as the probability of successful communication rather than a simple binary on/off relation.

Link/neighbor selection: The limited memory in the mote hardware prevents a node from holding state for all its links. Link selection determines the set of neighbors in a node's routing table.

Distance estimation: Recall that our BVR algorithm defines a node's coordinates as its distance in hops to a set of beacons. We describe how we define the hop distance from a node to a beacon when individual links are themselves defined in terms of a quality estimate.

Route selection: This addresses how a node forwards packets in the face of lossy links.

Each of the above is a research problem in itself (see [32, 33] for a detailed exploration of some of these issues); while our implementation makes what we believe are sound choices for each, a comprehensive exploration of the design space for each individual component is beyond the scope of this paper. We describe our solutions to each of the above problems and present the results of our system evaluation in the following section.

Currently, our prototype sets the number of routing beacons equal to the total number of beacons ($k = r$) and does not implement the successive dropping of beacons in computing distances for greedy forwarding (*i.e.*, a node that cannot make greedy progress using all available beacons switches directly to fallback mode). We also do not implement the on-demand neighbor acquisition described in the previous section. If anything, these simplifications can only degrade performance relative to our earlier simulation results.

5.1 Link Estimation and Selection

Estimating the qualities of the links to and from a node is critical to the implementation of BVR as this affects the estimated distance from beacons as well as routing decisions. For example, consider a node that on occasion hears a message directly from a beacon over a low quality link. If, based on these sporadic receptions, the node were to set its distance from the beacon to be one hop then that would have the undesired effect of drawing in traffic over the low quality link.

We implemented a passive link estimator, based on the work by Woo *et al.*[32]. We tag all outgoing packets with a sequence number, such that the receiving nodes can estimate the fraction of packets that are lost from each source. We collect statistics in successive time windows, and the estimation is derived from an exponentially weighted moving average of the quality over time. This estimates the quality of *incoming* links. To accommodate link asymmetry, every node periodically transmits its current list of incoming link qualities. It is aided by the fact that nodes transmit at a minimum rate, making estimation more reliable: in BVR, nodes periodically broadcast "hello" messages used to announce coordinates and maintain the beacon trees. The link estimator is also responsible for detecting "dead" neighbors, and to keep a table with the best quality links.

Because motes have limited memory, a node may not be able to (or may not want to devote the memory needed to) hold state for all the nodes it might hear from. Hence on the one hand we want a node to hold state for its highest quality links but on the other hand the node does not have the memory resources necessary to estimate the quality of all its links. To tackle this problem we use a scheme that guarantees that a node's link will store a set of neighbors with quality above a given low quality replacement threshold L. When a node is first inserted in the link table it is subject to a *probation* period. We set the probation period to be such that the link estimator would have converged to within 10% of the stable quality of the link. An entry in the link table cannot be replaced unless it is past probation and has a link quality below the replacement threshold. In our prototype, we use a link table size n of 18 neighbors, and set L to 20%.

5.2 Distance Estimation

Every node in BVR maintains two key pieces of information: (1) its distance in hops to the root beacons and (2) the positions of the node's immediate neighbors. The only control traffic for maintaining both consists of periodic local neighbor exchanges, in which nodes advertise their distance to each beacon, in the style of distance-vector routing algorithms. A beacon's periodic announcement includes a sequence number that is incremented at every interval. Through periodic neighbor exchanges, nodes build a reverse path tree to every beacon. A node maintains the highest sequence number and a parent along the tree to every beacon. These combined can eliminate count-to-infinity problems, loops, and allows for the detection of dead beacons.

Central to BVR is a node's distance in hops to each beacon. In the presence of lossy links, it is important to avoid using long and unreliable links, which can give the false impression of a low hopcount to the root. To this effect, nodes determine their distance from the beacon by choosing parents that minimize the expected number of transmissions (ETX) to the root [32] along the reverse path. The ETX for one link with forward and reverse transmission success probability p_f and p_r is $\frac{1}{(p_f \times p_r)}$, and the ETX to the root is obtained incrementally by adding the ETX for each link. A node's distance is the number of hops along such path. We use some hysteresis when selecting parents with different hopcounts to increase the stability of the coordinates.

5.3 Route Selection

When selecting the next hop in forwarding a message, our BVR prototype takes into account both the progress in the distance function and the quality of the links. Usu-

Link Estimator	
Size of Table	18
Expiration	5 succ. windows
Replacement Thresh.	20% quality
Reverse Link Info Period	17.5s $\pm$ 50% (jittered)
Update Link Period	30 (fixed)
Exponential Average	smoothing constant 40%
BVRState	
Position Broadcast T	10s $\pm$ 50% (uniform)

Table 5: Parameters used in the experiments on both the *Office-Net* and *Univ-Net* testbeds

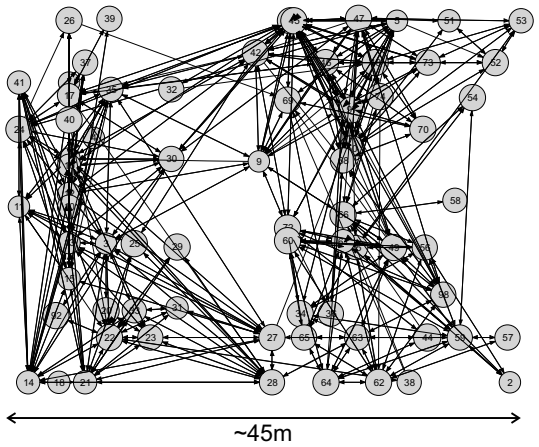

Figure 6: Neighborhood graph as determined by the neighbor tables of motes. Each node is shown with its ID. The positions of the nodes are to scale.

ally, the nodes that make the most progress are also further away, and present poor link quality. We order all links that make some progress towards the destination by *expected progress*: the product of the bidirectional link quality and the actual progress in the distance function. This is analogous to the $PRR \times Distance$ metric from [28], found to be optimal for geographic routing. When sending a message, we use two optimizations for reliability. First, we use link level acknowledgments with up to five retransmissions. Second, if a transmissions fails despite the multiple retries, the node will try the other neighbors in decreasing order of their expected progress. Only when it has exhausted all possible next hop options will the node revert to fallback mode.

Table 5 summarizes our various parameter settings. We selected these based on both our own experience and those reported by Woo *et al.*[32] with the mote radios. We intend to achieve good tradeoff between maintaining freshness of the routing state, and the amount of control traffic generated. A back-of-the-envelope calculation based on our measured channel capacity indicates that our timer settings lead to control traffic of approximately 5% of the channel capacity. Fully understanding the generality of our parameter selection is beyond the scope of this paper and a topic we intend to explore. Nonetheless, because sensornet topologies (unlike most other networks) are dependent on so many deployment specific issues — interference and obstacles in the physical environment, number and layout of nodes, power settings, *etc.* — we do expect deployments to always involve some amount of a priori calibration to guide parameter selection.

6 Prototype Evaluation

This section presents the results of our experiments with the BVR prototype deployed over two testbeds. The first (Office-Net) consists of 42 `mica2dot` motes [10] in an indoor office environment of approximately 20x50m while the second (Univ-Net) is a testbed of about 74 `mica2dot` motes deployed across multiple student offices on a single floor of UC Berkeley's Computer Science building. In both testbeds, motes are connected to an Ethernet backchannel that we use for logging and driving the experiments. These testbeds are of moderate scale, with diameters of five and 7 hops, respectively, and hence do not truly stress BVR's scalability. Nonetheless, these deployments are an invaluable (if not the only!) means by which to test our algorithms under the non-uniform and time-varying radio characteristics that cannot be easily captured in simulation.

On both testbeds, we set parameters as described in the previous section. Our experiments consist of a setup phase of several minutes to allow the link estimations to converge, beacon trees to be constructed and nodes to discover their neighbors' positions. After this setup phase, we issue route commands from a PC to individual motes over the Ethernet backchannel.

In the reminder of this section we evaluate four main aspects of the BVR design:

Link Estimation: We validate that a node indeed selects high quality neighbors. This is important because, as has been reported, the details of link estimation are at once tricky and greatly impact performance. Moreover, because link quality is a function of environment, topology and traffic patterns, we could not just expect behavior identical to previous studies [32, 33].

Routing Performance: We evaluate BVR's success rate on two testbeds under increasing load. We find that the routing success rate is high (over 97%) when the network load is low, and degrades gracefully as the load increases.

Dynamics: We evaluate performance under both node and beacon failures and show that BVR sustains high performance even under high node failure rate.

Coordinate Stability: Because many applications may use node coordinates as addresses, it is important that these coordinates vary little in magnitude and over time. We find that BVR coordinates are quite stable.

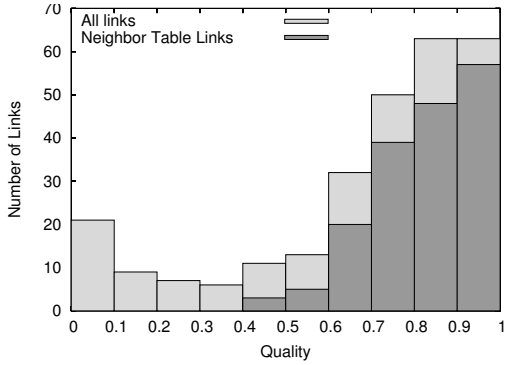

Figure 7: Histograms of the measured link qualities of all links and of the subset of these links chosen that are part of motes' routing tables. Notice how the latter are proportionately better quality.

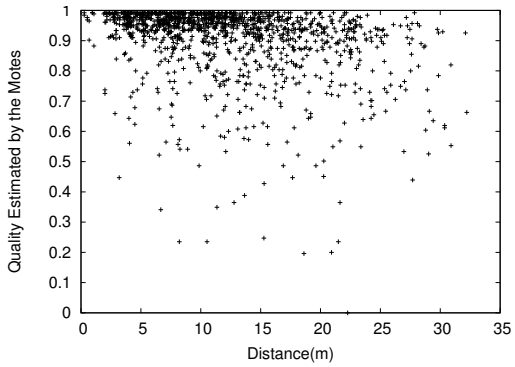

Figure 8: Link quality versus distance. Network connectivity is not always correlated with physical distance.

6.1 Link Estimation

We verify that BVR successfully estimates individual link qualities and selects high quality neighbor links over which to route. Our data also shows little correlation between distance and link quality. Since the evaluation of the link estimator does not require a large multi-hop network, we obtain our results from a subset of twenty three motes in our Office-Net environment. Based on the packets logged at each mote, we record the true quality of every link over which even a single packet was received. Figure 7 compares these measured link qualities to those of the subset of links selected by motes in their routing tables. Note how the fraction of neighbor links selected in each range of quality increases with quality, which confirms that nodes choose links with comparatively good qualities to be part of their coordinate tables.

We also examined the network-wide connectivity in our testbeds. Figure 6 shows a snapshot of the network, drawn to scale, and the connectivity as determined by the neighbor tables at each mote on the 74 node Univ-Net testbed. We see that network connectivity is frequently not congruent with physical distance (*e.g.*, mote pairs 32-30, 32-9, 26-37, 35-27, 54-59). We also note the existence of short but asymmetric links (motes 24-26).

For the same testbed, Figure 8 shows the relation between link quality and physical distance between pairs of nodes that *are* neighbors (as determined by BVR's link and neighbor selection algorithms). While BVR select predominantly high quality neighbors, these are quite frequently not physically close; in fact, a fair number of neighbors are more than halfway across the network. Note that these observations contradict the circular radio assumptions made by typical geographic routing algorithms and lend credibility to the need for connectivity-derived coordinates.

6.2 Routing Performance

In the following experiments, after the setup phase, each mote periodically attempts to route to a random destination mote. We present BVR routing performance in terms of successful packet delivery under increasing routing load. In these tests we do not experiment with beacon selection, and just preconfigure 5 well spread nodes to act as beacons. In the following section we evaluate our implementation of beacon selection using the low-level mote simulator TOSSIM [18].

Figure 9 present results for the Office-Net and Univ-Net testbeds. The graphs show: (1) the overall success rate measured as the fraction of routes that arrived at the target destination, (2) the fraction of routes that required scoped flooding, (3) the fraction of routes that failed due to contention drops where contention drops are packets that were dropped due to a lack of sending buffers along the internal send path in the mote network stack and (4) failures which are routes that failed despite the multiple retries; *i.e.*, the message was repeatedly sent out over the channel but no acknowledgments were received. The graph also plots the aggregate network route request rate over time with the scale on the right hand Y axis. Our tests start (after the setup phase) with a rate of one route request per second for a period of approximately one hour; after this we increase the route request rate every 400 seconds up to a maximum rate of approximately 8 routes/second.

On Office-Net, BVR achieved an average success rate (greedy or flood) of 99.9% until a load of about 8.8 requests/second, at which point we start seeing a small number of contention drops. 1.2% of all route requests in this period resulted in scoped flooding (with an average scope of 2 hops), and less than 0.1% were contention drops. Similarly for Univ-Net, the average success rate was 98.5%. 5.5% of all routes required scoped flooding (with again an average scope of 2 hops), there were no contention drops, and 1.15% of routes failed due to persistent loss. We repeated the above tests with a larger number of beacons and recorded similarly high success

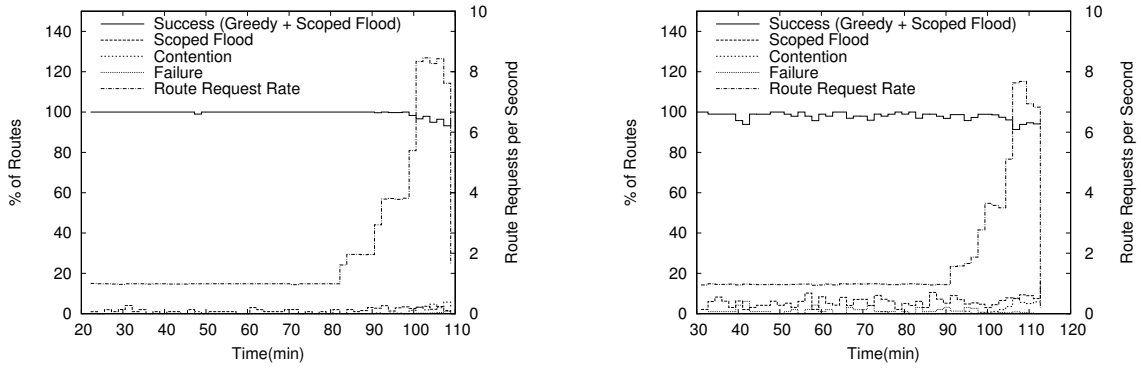

Figure 9: Results of routing tests under increasing routing load, for *Office-Net* on the left and *Univ-Net* on the right. Rate on the right hand Y axis is the aggregate number of route requests issued per second in the network.

rates; due to space considerations, we do not present these here. These experiments indicate that our BVR implementation works correctly in a real deployment, and can sustain a significant workload of routing messages.

6.3 Node Dynamics

Sensor nodes are vulnerable to temporary or permanent failures due to depleted energy resources, or damage from weather conditions. We test BVR's resilience to such failures using both TOSSIM and real testbed experimentation. We show that BVR can recover from both node and beacon failures and sustain good routing performance without incurring high overhead. We first evaluate BVR's robustness to non-beacon node failure using the Office-Net testbed and then evaluate robustness to beacon failure in TOSSIM.

6.3.1 Node Failure

To verify BVR's robustness to node failure in a real deployment, we ran tests on the Office-Net testbed with artificially induced node failures. The setup for this is identical to that in Section 6.2 except that we maintain a query load of one route/second and, after a warming period, repeatedly kill one random (non-beacon) mote every five minutes until all but the beacons nodes have been killed. Note that, once dead, we never resurrect a mote; this failure model is more realistic for sensor networks where the predominant cause of failure is battery exhaustion and not (as on the Internet) node reboots or disconnections [30]. Figure 10 plots the success rate (along with the number of live motes) over time. We see that BVR is extremely resilient to random node failure. The routing success rate remains mostly high until well over 80% of the motes are killed, at which point the success rate drops. Closer examination of the logs revealed that while node failures do lead to occasional dips in success rate, BVR quickly recovers. This behavior stems from the redundancy in the coordinate system, the route selection mechanisms, and the adaptability of the coordinates to the topology changes.

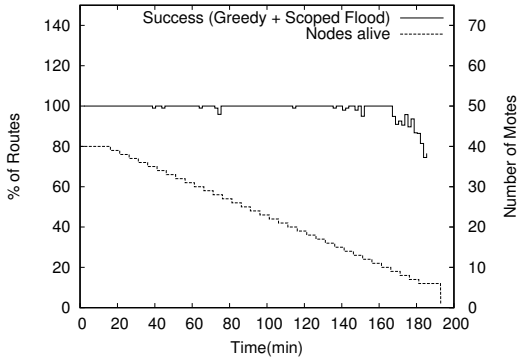

Figure 10: *Office-Net* Success rate (left Y axis) and number of live motes (right Y axis) over time. BVR maintains high success rates under increasing node failure.

6.3.2 Beacon Failure

Section 3 discussed the need to maintain a reasonable number of beacon nodes in the network. Our BVR implementation includes Algorithm 2 which we tested using TOSSIM. Our TOSSIM experiments choose a baseline configuration of 100 motes with 8 beacons and expected node degree of 12. We use TOSSIM's lossy link generator, which is itself based on empirical data and includes lossy and asymmetric connectivity. After a 30 minute setup phase, we initiate a constant rate of one route per second between random node pairs and kill a randomly selected beacon node at increasing rates. Success rates are computed in 100 second time windows under traffic load of 1 route request per second.

Figure 11 shows a typical simulation run. We see that routing performance does not degrade with occasional beacon failures, and tolerates high failure rates reasonably well. This is largely because BVR routes well with even a partial beacon vector set. Residual beacon vectors from recently deceased beacons also serve as hints for packet forwarding. Closer examination of the test log shows that the convergence time of the beacon replenishment is fast, and dependent as expected on the network diameter and frequency of neighbor exchanges. Candi-

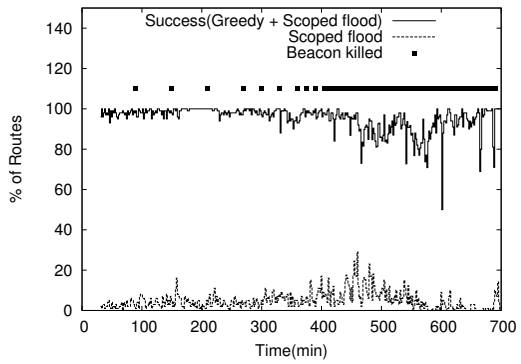

Figure 11: Resilience to beacon failure (TOSSIM): Randomly chosen beacons fail at increasing rates up to a maximum rate of 1/10 per min. Success rates are computed in 100 second windows.

date beacons are efficiently suppressed and hence we observed no significant communication overhead.

6.4 Coordinate Stability

Our results so far have shown that BVR generates good coordinates in that they correctly guide routes towards a target destination. Coordinates stability is important for routing performance, especially for applications that require a location database such as that described in Section 3. Not only may routing to outdated coordinates lead to routing failures, but constant changes can generate heavy update and lookup traffic close to the beacons.

In Figure 12 we look at the number of nodes with changes in coordinates per fixed 100-second intervals for a run in *Office-Net*, with a load of one route per second. This approximates the minimum aggregate update traffic that would be seen in the network if motes were to update at most once every such period, when their coordinates changed. From the graph we see that there are more nodes with changes in the beginning, as the link estimators are stabilizing, and then there is a reduction. The average number of motes with changes per slot is 0.95, and the 95th percentile is 9. We also looked at 10 second intervals, and the average number of motes with changes is 0.13 per interval, which is relatively consistent. In terms of number of changes, during this period of 100 minutes, 90% of the motes had 5 or fewer changes in coordinates. The results we saw in the *Univ-Net* testbed are similar. Figure 13 plots the distribution of the magnitude of individual coordinate changes over all coordinate changes. Magnitude here is simply the change in a node's distance to a beacon; *i.e.*, a change in distance to a beacon from 5 hops to 3 hops would be counted as a magnitude of 2. We see that change, when it occurs, is small: for both testbeds, in this case, at least 80% of the changes were of 2 or less hops. These results suggest that BVR can be used as a stable routing solution that is scalable, robust, and does not unduly load beacons.

7 Conclusions and Future Work

Beacon Vector Routing is a new approach to achieving scalable point-to-point routing in wireless sensornets. Its main advantages are its simplicity, making it easy to implement on resource constrained nodes like motes, and resilience, in that we build no large-scale structures. In fact, the periodic flooding from the beacons means that no matter what failures have occurred, the entire state can be rebuilt after one refresh interval. Our simulation results show that BVR achieves good performance in a wide range of settings, at times significantly exceeding that of geographic routing. Our implementation results suggest that BVR can withstand a testbed environment and thus might be suitable for real deployments.

However, we are at the very early stages of our investigation. We need to better understand how BVR's performance is linked to radio stability, the generality of our parameter selection as well as more rigorous approaches to tuning these parameters. Most importantly however, we have not yet implemented any applications on top of BVR, so we don't yet know if it provides a suitably stable substrate on which to build. All of these items represent future work.

References

[1] BOSE, P., MORIN, P., STOJMENOVIC, I., AND URRUTIA, J. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks 7*, 6 (2001), 609–616.

[2] CAO, Q., AND ABDELZAHER, T. A scalable logical coordinates framework for routing in wireless sensor networks. In *IEEE Real-time Systems Symposium* (December 2004).

[3] COUTO, D. D., AGUAYO, D., CHAMBERS, B., AND MORRIS, R. Performance of multihop wireless networks: Shortest path is not enough. In *Proceedings of HotNets* (Oct. 2002).

[4] DEMIRBAS, M., ARORA, A., AND GOUDA, M. A pursuer-evader game for sensor networks. In *Proceedings of the Sixth Symposium on Self-Stabilizing Systems* (2003), pp. 1–16.

[5] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of SIGCOMM* (1995), ACM Press, pp. 342–356.

[6] GANESAN, D., ESTRIN, D., AND HEIDEMANN, J. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? In *Proceedings of the ACM HotNets* (October 2002), ACM, pp. 143–148.

[7] GIROD, L., STATHOPOULOS, T., RAMANATHAN, N., ELSON, J., ESTRIN, D., OSTERWEIL, E., AND SCHOELLHAMMER, T. EmStar: An environment for developing wireless embedded systems software. In *Proceedings of the Second SenSys* (Nov. 2004), ACM Press.

[8] GREENSTEIN, B., ESTRIN, D., GOVINDAN, R., RATNASAMY, S., AND SHENKER, S. DIFS: A distributed index for features in sensor networks. In *Proceedings of First IEEE WSNA* (May 2003).

[9] HAMILTON, M., ALLEN, M., ESTRIN, D., ROTTENBERRY, J., RUNDEL, P., SRIVASTAVA, M., AND SOATTO, S. Extensible sensing system: An advanced network design for microclimate sensing, June 2003.

[10] HILL, J., AND CULLER, D. Mica: a wireless platform for deeply embedded networks. *IEEE Micro 22*, 6 (November 2002), 12–24.

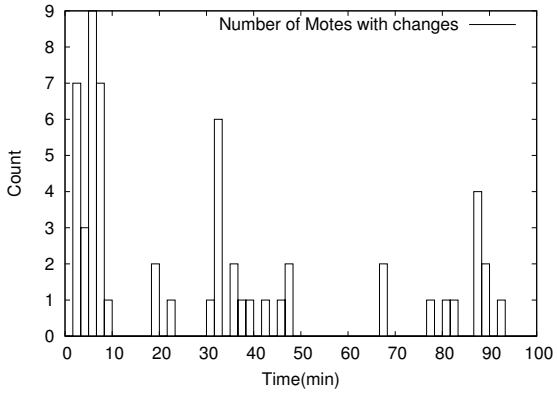

Figure 12: Number of motes with at least one coordinate change per slot of 100 seconds. This approximates the update traffic the location database would see, assuming a minimum update period of 100 seconds per mote.

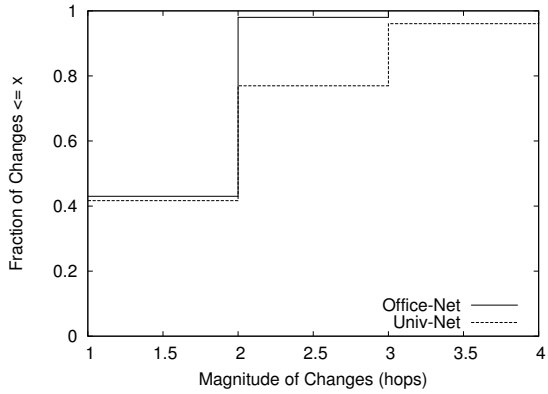

Figure 13: Distribution of the magnitude of individual coordinate changes over all coordinate changes seen across all nodes over test durations of 60 minutes on the Office-Net and Univ-Net testbeds.

[11] HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. System architecture directions for networked sensors. In *Proceedings of the ASPLOS* (2000), ACM Press, pp. 93–104.

[12] INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. Directed Diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual MOBICOM* (2000), ACM Press, pp. 56–67.

[13] JOHNSON, D. B., AND MALTZ, D. A. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, Imielinski and Korth, Eds., vol. 353. Kluwer Academic Publishers, 1996.

[14] KARGER, D. R., LEHMAN, E., LEIGHTON, T., LEVINE, M., LEWIN, D., AND PANIGRAHY, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. 29th ACM STOC* (May 1997), pp. 654–663.

[15] KARP, B., AND KUNG, H. T. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual MOBICOM* (2000), ACM Press, pp. 243–254.

[16] KUHN, F., WATTENHOFER, R., ZHANG, Y., , AND ZOLLINGER, A. Geometric ad-hoc routing: Of theory and practice. In *22nd ACM PODC* (2003).

[17] KUMAR, S., ALAETTINGLU, C., AND ESTRIN, D. Scalable object-tracking through unattended techniques (scout). In *Proceedings of the 2000 International Conference on Network Protocols* (2000), IEEE Computer Society, p. 253.

[18] LEVIS, P., LEE, N., WELSH, M., AND CULLER, D. Tossim: accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the First SenSys* (2003), ACM Press, pp. 126–137.

[19] LEVIS, P., MADDEN, S., GAY, D., POLASTRE, J., SZEWCZYK, R., WOO, A., BREWER, E., AND CULLER, D. The emergence of networking abstractions and techniques in tinyos. In *Proceedings of the First USENIX/ACM NSDI* (March 2004).

[20] LI, X., KIM, Y. J., GOVINDAN, R., AND HONG, W. Multidimensional range queries in sensor networks. In *Proceedings of the First SenSys* (2003), ACM Press, pp. 63–75.

[21] MADDEN, S. *The design and evaluation of a query processing architecture for sensor networks.* PhD thesis, UC Berkeley, 2003.

[22] MADDEN, S., FRANKLIN, M., HELLERSTEIN, J., AND HONG, W. TAG: a tiny aggregation service for ad hoc sensor networks. In *OSDI* (2002).

[23] MAINWARING, A., POLASTRE, J., SZEWCZYK, R., CULLER, D., AND ANDERSON, J. Wireless sensor networks for habitat monitoring. In *Proceedings of ACM WSNA* (Sept. 2002).

[24] NEWSOME, J., AND SONG, D. Gem: Graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the First SenSys* (2003), ACM Press, pp. 76–88.

[25] RAO, A., RATNASAMY, S., PAPADIMITRIOU, C., SHENKER, S., AND STOICA, I. Geographic routing without location information. In *Proceedings of the 9th Annual MOBICOM* (2003), ACM Press, pp. 96–108.

[26] ROYER, E. M., AND TOH, C.-K. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications 6* (April 1999), 46–55.

[27] SAVVIDES, A., HAN, C.-C., AND SRIVASTAVA, M. B. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Mobile Computing and Networking* (2001), pp. 166–179.

[28] SEADA, K., ZUNIGA, M., HELMY, A., AND KRISHNAMACHARI, B. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proceedings of the 2nd SenSys* (2004), ACM Press, pp. 108–121.

[29] SHENKER, S., RATNASAMY, S., KARP, B., GOVINDAN, R., AND ESTRIN, D. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev. 33*, 1 (2003), 137–142.

[30] SZEWCZYK, R., POLASTRE, J., MAINWARING, A., ANDERSON, J., AND CULLER, D. An analysis of a large scale habitat monitoring application. In *Proceedings of the Second SenSys* (2004), ACM Press.

[31] TSUCHIYA, P. F. The Landmark Hierarchy: a new hierarchy for routing in very large networks. In *ACM SIGCOMM* (1988), ACM Press, pp. 35–42.

[32] WOO, A., TONG, T., AND CULLER, D. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the First SenSys* (2003), ACM Press, pp. 14–27.

[33] ZHAO, J., AND GOVINDAN, R. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the First SenSys* (2003), ACM Press, pp. 1–13.

[34] ZHAO, J., GOVINDAN, R., AND ESTRIN, D. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of the IEEE SNPA* (May 2003).