

Beacon Vector Routing: Towards Scalable Point-to-Point Routing in Deeply Embedded Networks

Rodrigo Fonseca
University of California, Berkeley
Final report – CS 294-1

Fall 2003

Abstract

In this paper we study the problem of point-to-point routing in deeply embedded wireless networks, and propose a novel algorithm, Beacon Vector routing, that is scalable, works with local information, and does not depend on geographic or pre-configured information. Our approach uses a greedy algorithm similar to Geographic Routing for choosing the next-hop when forwarding. A coordinate system is build, based on network connectivity, in which each node knows its distance (in network hops) to a set of reference nodes. When given a destination (a vector of distances), a node chooses as its next-hop the neighbor with the ‘closest’ distances to the reference nodes.

We describe our prototype implementation of BV Routing in TinyOS, and some measurements both in simulation and on real hardware. We also discuss alternative implementation options, and future steps.

1 Introduction

In this paper we present a novel approach for achieving point to point multihop routing in wireless embedded networks. Given the resource constrained nature of these environments in terms of memory and energy, a useful routing mechanism should be able to achieve low stretch routes, be scalable with respect to network density, number of nodes, and patterns of communication, and be robust to variations in connectivity, node failures, and obstacles in general. It should also be simple, not requiring too complex setup phases or too complex computation on low end devices.

Many algorithms have been developed to provide point to point multihop routing in wireless networks, specially in the MANET space [9] over the last few years, but many of them have problems with scalability. One of the difficulties in the space of ad-hoc networking is the fact that there is no preset naming structure that can be associated in a somewhat fixed manner with the position within the network. This precludes the use of naming aggregation techniques that make Internet routing scalable.

On the other hand, most multihop routing algorithms that have been developed and used for wireless sensor networks focus on many-to-one or one-to-many routing, and are hard to extend for more general traffic patterns.

We want to avoid the use of on demand flood for route discovery: ideally, routing should be done with only local information. We also want to have small state requirements for routing: this should be on the order of our local neighborhood, i.e., those nodes within communication range. Geographic routing is a technique

that achieves these goals, with the requirement that the nodes know their current geographic positions for routing.

Our approach, Beacon Vector routing (BV routing), is similar in spirit to Geographic Routing: like that algorithm, nodes can route in a greedy fashion with information about the destination ‘location’ and about their immediate neighborhood. It can be cast as a gradient routing protocol, and thus seen as part of a broad class of algorithms which establish a gradient for routing that converges to the destination.

However, differently from Geographic routing, we assume that the nodes do not have geographic information. Instead, we label the nodes with ‘virtual positions’, which are derived from the network connectivity or topology instead of from geography. Routing is performed on these labels. The key for routing is that it is possible, given a destination position and the current position, to obtain a dissimilarity measure between the two that correlates with the network distance (in hops). When forwarding a message, a node can choose the neighbor whose position has the least dissimilarity with the destination, and send it to this node. The positions are determined relative to a set of reference points (which we call Root Beacons), as we describe in more detail later.

Our focus in this report is on implementation of a prototype of BV routing on TinyOS [2], the additional challenges that the environment poses, and the many directions for continuing the work. The rest of the document is organized as follows. We give a brief discussion of related work in the following section. Details of the algorithm are discussed in Section 3, including how the coordinate space is created and maintained, and how routing works. Specific details about the implementation of BV routing on TinyOS are given in Section 4. Section 5 presents our evaluation methodology and some initial results, and we present concluding remarks, and our next steps, in Section 6.

2 Related Work

There exists a substantial body of literature on routing algorithms in general, with algorithms that fit a broad range of applications and assumptions.

Most proposed point-to-point algorithms for ad-hoc wireless networks [9] rely on some type of flooding for route discovery, are found not to scale well with the number of flows or network size.

Geographic Routing [3] is perhaps the most scalable solution to the problem, as it uses a global frame of reference – the geographic positions of the nodes – to establish a gradient to each possible destination. Each node, when required to route a message to a given destination (in the coordinate space), forwards the message to the node which is closest, also in the coordinate space, to the destination. The algorithm has minimum state requirements, as nodes are only required to know about their neighbors’ positions to route, and as such has been found to scale very well. There are two drawbacks, however, when applying Geographic Routing to wireless embedded networks: the first is that the geographic locations may not be available to nodes, and the second is that in the low-power nature of the radio may lead to a weaker correlation between geographic distance and network connectivity. Geographic routing has a lower performance when presented with either low density or substantial obstacles.

As an attempt to find an answer to these problems, algorithms have been proposed [7, 5] that create a virtual coordinate system directly from the connectivity of the network, and then route using variants of Geographic Routing on the created coordinate systems. Our work derives from the one in [7], as an attempt to simplify the initial setup phase of that algorithm.

Other routing algorithms try to create routing information from the network connectivity. Landmark routing[12] is one such algorithm, in which a number of reference points with varying scope form a hierarchical structure that allows routing. There are a few landmarks that are known to the entire network, and

progressively more landmarks that have progressively smaller scope. An address in this network consists of the closest landmark to a node at each landmark level, and when routing a node always sends the message up the tree of the smallest scope landmark in the address that it knows about. Our approach also uses nodes as reference points, but the difference is that we do not route directly towards the reference points, but rather trying to directly decrease an estimate of the distance to the destination given by the information of combined reference points. It is an interesting exercise to compare the benefits and pitfalls of both approaches, which we leave for future work.

The research on multihop routing in the wireless embedded networks field [14, 15] has focused mainly on many-to-one or one-to-many routing, with assumptions on the traffic pattern that make it hard if not impractical to extend the results directly to arbitrary point-to-point routing. However, many of the problems analyzed in such research are the same as the ones we face, such as forms of tree formation and path reliability estimation, neighborhood discovery and management, link quality estimation, and retransmission and scheduling issues.

Finally, we ought to mention work done for proximity selection on the Internet space, in which an important problem is how to select a server, or a proxy server, which is close to the user. In [6], the authors evaluate how different ways of attributing coordinates to servers on the Internet correlate to their true distances (in terms of latency). One of the approaches they use is that of establishing coordinates by determining the distances from several reference points. This notion is also employed in [11], which also use distance estimates from several reference points to estimate the distance. They however use techniques of dimensionality reduction to create a smaller number ‘virtual landmarks’ that still are able to yield good predictions. We also use the technique of using distances from a set of reference points to predict the true distance, but to the best of our knowledge this work is the first to use such estimates for routing.

3 Algorithm

In this section we describe the routing algorithm that we use, and provide some intuition on how it works. The goal with BV Routing is to achieve the benefits in terms of scalability that Geographic Routing provides, except we don’t assume that nodes have their location information.

In a high level, the algorithm can be described as follows: we assign coordinates to the nodes, in such a way that a dissimilarity function between these coordinates correlates with the true network distance. Routing is performed in a greedy fashion, with nodes at each step trying to make progress by choosing a neighbor that decreases the dissimilarity function with the destination. This greedy routing is similar in nature to the greedy routing in geographic routing [3]. However, the coordinate system that we use is completely derived from the network topology itself. There are situations in which the greedy routing procedure cannot make progress, and we outline some of the reasons below. In this case, the nodes enter a ‘fallback’ mode to try to recover from the point where routing was stuck. We now move on to describing the algorithm in more detail.

3.1 Coordinate Establishment and Routing

We first describe the coordinate system that we use for routing and how it can be established. A subset of the nodes is selected as ‘root beacons’. These will act as reference points for routing. Each node learns, from the network, the distance in hops to each of the beacons. This is done by a process of flooding initiated by each root beacon. Each root beacon broadcasts a ROOT message, with its *id*, which is heard by its neighbors. The neighbors, in turn, rebroadcast this message, increasing the *hopcount* for that root, in an recursive process that reaches the entire network. This in effect forms a spanning tree of the network, rooted

at the beacon. As we shall see below, it is interesting for each node to keep track of its parent node in each root beacon tree, which is useful in the fallback routing mode.

We call the ordered set of distances between the node and the beacons its *position*. Whenever we refer to the hop distances between two nodes, we are referring to the shortest path distance between these nodes. This is important as the shortest path distance metric respects the triangle inequality. If there are n beacons, the position of a node k is given by the n -tuple

$$\mathcal{P}(k) = \langle B_1 : p_1, B_2 : p_2, \dots, B_n : p_n \rangle.$$

This is a form of a *Lipschitz embedding* [11], and it is based on these positions, or coordinates, of the nodes that routing takes place.

Next, we define a dissimilarity metric δ between the positions of two nodes p and q according to the following equation:

$$\delta(p, q) = \sum_{i=1}^n \omega_i |p_i - q_i|, \quad (1)$$

i.e., as a weighted sum of the absolute differences of the distances of the two nodes to respective beacons. We postpone the description of the ω factors to Section 3.2. This metric has some properties of a distance, but is not necessarily symmetric, depending on the ω factors. It is 0 if and only if $\mathcal{P}(p) = \mathcal{P}(q)$, and it is non-negative. Intuitively, two nodes that are close to each other in the topology will have similar distances to the root beacons. Each node in the network also learns about the positions of each of its neighbors, which is a minimum requirement for routing. In Figure 1(a) we show a simple example network, with three beacons B_1, B_2 , and B_3 , and all nodes labeled with their positions.

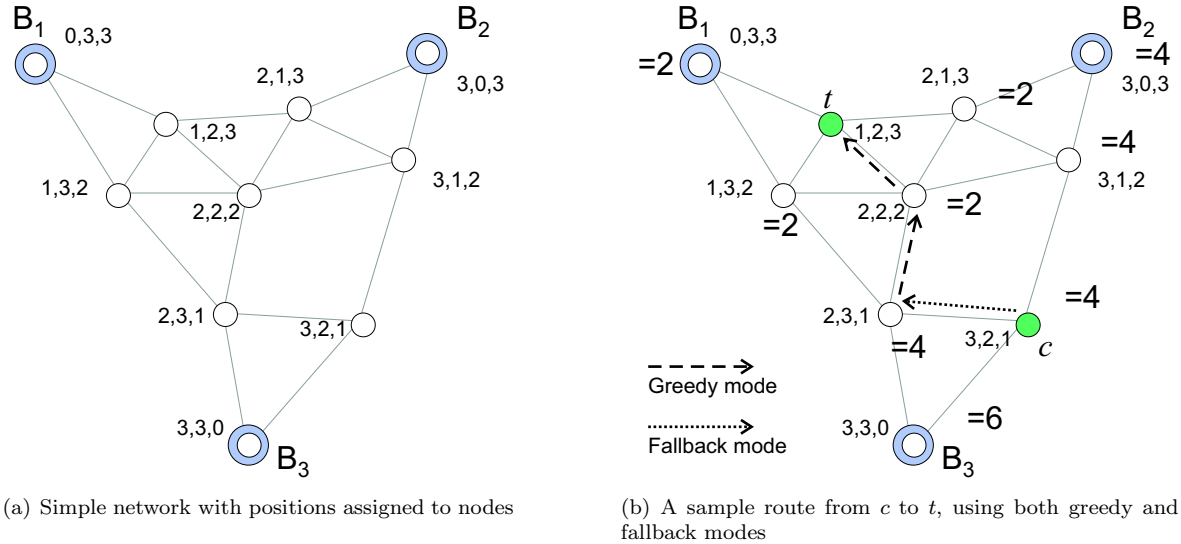


Figure 1: Routing in the geometric space from c to t

Greedy Routing Assuming that each node knows its position and those of its neighbors, we now describe the routing part of the algorithm. We assume that we are given packets to route, and that these packets have at least two fields in the header: a destination position $\mathcal{P}(t)$, and δ_{min} , the minimum value of the

Packet header fields	Description
packet.destination	position $\mathcal{P}(t)$ of the destination
packet.t	the unique id of the destination t
packet. δ_{min}	the smallest dissimilarity (in greedy mode) seen by the packet so far
packet.fallback	whether the packet is in fallback mode or not

Table 1: Packet header fields to support BV Routing

dissimilarity metric that the packet has seen so far. It is also useful to store in the packet id_{dest} , the unique node id of the destination, as we shall see below (the fields in the packet header used for routing are described in Table 3.1). When forwarding the message, a node c chooses among its neighbors the node n such that

$$n = \underset{n_i \in N(c)}{\operatorname{argmin}} \{\delta(n_i, t)\}.$$

If $\delta(n, t) < \delta_{min}$ from the packet, then node c stores $\delta(n, t)$ in the packet as δ_{min} and forwards the packet to node n .

Fallback mode In the case in which no neighbor is found to make progress in the dissimilarity metric, the node enters the fallback routing mode. First, a flag is set in the packet ($fallback = TRUE$). Then, the node routes the packet towards the root beacon which is closest to the destination. This beacon is readily determined from the position of the destination in the packet. To accomplish this, we use the neighbor that is the parent of the current node in the corresponding root beacon tree. The nodes are able to route packets towards each of the root beacons, because at least one of its neighbors must be closer to the root itself (the node’s parent in the tree)¹.

At each hop of the route, when in fallback mode, the node tries to resume greedy routing. Recall that in the packet we store δ_{min} . Before forwarding the packet in the fallback mode, the node verifies whether one of the neighbors n has $\delta(n, t) < \delta_{min}$. If so, the packet *fallback* flag is reset, and the packet is forwarded normally to the node n .

In Figure 1(b), we show a route in the simple network, in which node c wants to send a packet to node t .

Scoped Flood It may be the case that a packet does not resume greedy routing, and ultimately reaches the beacon closest to the destination while in fallback mode. When this happens, we note that to guarantee that the algorithm is loop-free, the root beacon cannot try to forward the packet in the greedy mode even if that is possible. At this point, the only information that this root beacon has about the route to the destination is the distance in hops from itself to the destination, which can be determined from the destination’s position in the packet. The root beacon then initiates a scoped flood in the network, broadcasting the packet to all of its neighbors, with a *time-to-live* set to the distance to the destination. This final step of the algorithm is what guarantees its correctness. Of course, since flooding is an inherently expensive operation, we want to minimize the frequency with which this step is performed, and also the scope of the flood. The former is one of the metrics we choose to evaluate the algorithm, while the second is guaranteed to be minimal by the fact that we choose the root beacon which is closest to the destination.

Table 3.1 lists the fields that are present in the packet header to allow for BV Routing. In Algorithm 1, we list pseudo-code for the routing algorithm as we described above.

¹For this assumption to be true, the links should be bidirectional. We may enforce this by having a node ignore messages it hears from a link known to be inbound only.

Algorithm 1 BV Routing forwarding algorithm

BVR_FORWARD(current node c , packet P)

```
{greedy forwarding}
 $n \leftarrow \operatorname{argmin}_{n_i \in N(c)} \{\delta(n_i, P.t)\}$ 
if  $\delta(n, t) < P.\delta_{min}$  then
  if  $P.\text{fallback}$  then
     $P.\text{fallback} \leftarrow \text{FALSE}$ 
  return  $n$ 
{fallback mode}
 $P.\text{fallback} \leftarrow \text{TRUE}$ 
 $\text{fallback\_beacon} \leftarrow$  beacon with smallest distance in  $P.\text{destination}$ 
if  $\text{fallback\_beacon} \neq c$  then
  return PARENT( $\text{fallback\_beacon}$ )
{scoped flood}
broadcast  $P$  with scope  $P.\text{destination}[\text{fallback\_beacon}]$ 
```

3.2 Discussion

Having described the algorithm, we now discuss some of its aspects in more detail, and provide some intuition as to how it works.

We first note the relation between the difference terms in Equation 1 and the true distance between the source and the destination. Recall that p_i is the distance between node p and beacon B_i , and let pq be the distance between points p and q . It is easy to see, directly from the triangle inequality, that

$$pq \geq |p_i - q_i|,$$

i.e., that the absolute difference provides a lower bound for the true distance, which can also be viewed as an estimate of the true distance. The bound is tight if and only if one of the points lies on the path between the beacon and the other point. In particular, it is tight if one of the nodes is the beacon. On the other hand, the bound is worse when the beacon is equidistant from both points. If we take this view, we can see that $\delta(p, q)$, defined in Equation 1, is proportional to the average of the estimates given by each beacon, and in fact, if scaled properly, is itself also a lower bound for the true distance pq . Another way to look at the algorithm is as an approximation of Distance Vector routing (DV). If, in an extreme case, all nodes are made to be beacons, the algorithm, with some slight modifications, reduces to DV routing.

In order to better understand the progress of routing, it is worthwhile to use a geometric analog of the algorithm, in which we use points instead of nodes, and euclidean distances instead of hop-count distances. This is a valid comparison to make in many senses, because both distance metrics (euclidean distance in a Cartesian space, and shortest path hop distance, in the case of a network graph), respect the triangle inequality.

Figure 2 illustrates a step of the algorithm in the geometric space, for two different settings, assuming we have only one beacon, B_0 . c is the current node which must make a routing decision, and t is the destination. We assume that c has a communication range given by the circle with radius r in the figure. With respect to one beacon B_0 , the set of nodes n that have minimum $\delta(n, t)$ lie on the ball centered at B_0 with radius B_0t . c has to choose a neighbor c' that minimizes $\delta(c', t)$, which is shown in the figure also. When this process is repeated iteratively, the packet will eventually reach a node t' , which is the closest node to c that lies on the ball B_0t . The two settings in Figure 2 have one important distinction: whether the beacon is closer to t or c . We note that in the former, represented in Figure 2(a), the successive points in the route are progressively closer to the beacon, while in part (b), the points are progressively farther from the beacon, and that in both they converge to a point in the ball B_0t .

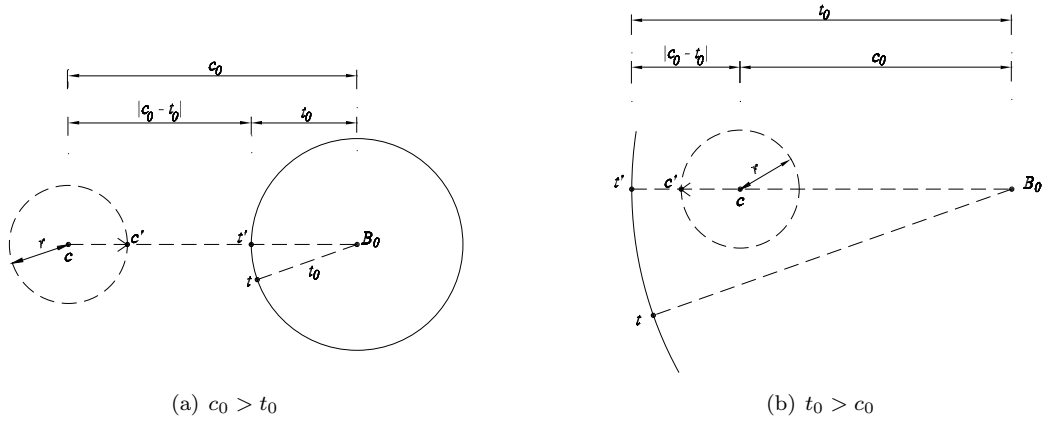


Figure 2: Routing in the geometric space from c to t

We now come back to the discussion of the missing piece in the definition of the dissimilarity metric (Equation 1), the ω_i factors. We observed that the beacons that ‘pull’ in the routing, i.e., the beacons that are closer to the destination than to the current node, are better for routing than those that ‘push’. Some intuition can be gained if we look at Figure 3. In these diagrams, a packet is being routed from node c to the node t . We only consider one root beacon as a reference. In this Figure, we note that the value of $|Bc - Bt|$ is the same for both (a) and (b), but the sign of the difference itself is not. If we observe the point t' reached by routing using only this beacon, as in the previous figure, we see that in the case in which the beacon was closer to c than to t , the error is much larger than in the other case. This suggests that when making routing decisions, priority should be given to the beacons that are closer to the destination than to the current node, i.e., to those beacons for which $Bc - Bt > 0$. This is exactly the motivation for the ω_i factors, defined as follows:

$$\omega_i = \begin{cases} W & \text{if } p_i - q_i > 0 \\ 1 & \text{if } p_i - q_i \leq 0 \end{cases} \quad (2)$$

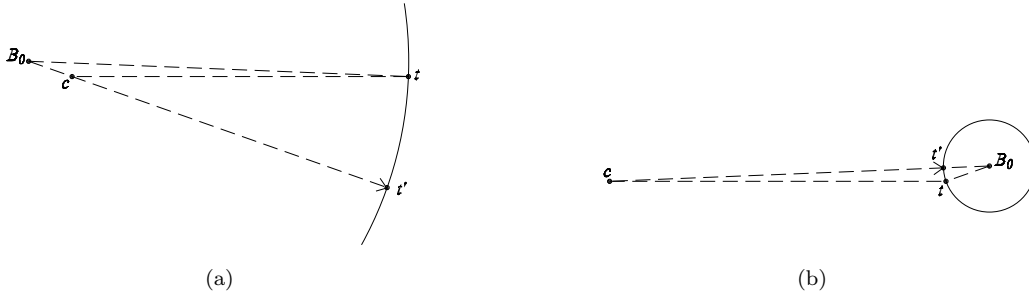


Figure 3: Routing on only one beacon. These two different beacons in relation to the same pair of source and destination for a route illustrate that even though the absolute difference of the distances to the beacon is the same in both cases, and that routing in this case tends to decrease the distance between the two nodes, the beacon that is closer to the destination (‘pulling’), leads to a much better endpoint.

In our evaluation and prototype, we use $W = 10$, which was found to give good results. The effect of having this different weights for the contributions of the different beacons is that the route taken gives priority to getting inside of the balls defined by all beacons.

The last point we discuss is two cases when greedy routing fails. While they do not represent all cases in which there are local minima in the routing gradient to a destination, they provide some insight as to when this may happen. We observed that sometimes neighbors can have the same measure of dissimilarity to a target, even though their positions are different. This happens because there is improvement in the difference in relation to a subset of the beacons, while the opposite happens with the difference respective to other beacons. This is exactly what happens in Figure 1(b), in which node c has no neighbor that improves $\delta(*, t)$. In this case, one hop in fallback mode is sufficient to recover to greedy routing.

Another type of problem may arise due to symmetries in the coordinate space, which are due to a combination of beacon placement and network topology. This can be seen in the geometric space, for example, when we have only two beacons. For each destination point t , with the exception of any point in the line that connects both beacons, there will be an image point t' in the other intersection of the two circles centered at the beacons that go through t . Nodes with the same coordinates, but very close in the network, do not present much of a problem, because they can easily learn about each other with purely local communication. The problem is more significant when nodes with the same coordinates are not in the same neighborhood. Adding beacons alleviates this problem, and reduces the chance of nodes in the network having very similar positions and not being close to each other.

3.3 Location

The algorithm, as described so far, assumes that the originating node knows the coordinates of the intended destination. Depending on the application, it may be necessary to map node identities to coordinates, in order to be able to reach a specific node. We identify two mechanisms to achieve this functionality.

The first one is based on consistent hashing and stores the mapping in the beacons. Each node in the network, except for possible transient inconsistencies, has information on how to reach each of the beacons, by routing to the neighbor from which it learned its distance from the given beacon. As a simple scheme, we propose the use of consistent hashing to provide a mapping $\mathcal{H} : nodeid \mapsto beaconid$, from node ids to the set of beacons, which can be computed by all nodes in the network. The location service then consists of two steps: each node k publishes its coordinates to its corresponding beacon $b_k = \mathcal{H}(k)$. When a node i first wants to route to k , it sends a coordinate request to the beacon b_k . Upon receiving a reply, it then routes to the received coordinates.

The alternative approach is to use the DHT structure described above to store the mappings. The difference from the above scheme is just where the information is stored. Each identifier is hashed to a set of random coordinates.

4 Implementation

We now describe our prototype implementation of BV Routing in TinyOS. To test the implementation, as we described in the introduction, we resort to running the same code (with small modifications), in the TinyOS Simulator – TOSSIM – and on Berkeley Mica motes.

Going from the high level simulator to a real implementation introduces many new problems that must be addressed. The challenges appear mainly due to the resource-constrained nature of the hardware, the difficulty in debugging code running on the motes, and on the complex interactions that the radio shows with the environment. The main issues that have to be addressed are how to acquire and store information about the neighborhood of a node, how to best estimate the quality of the links to and from these neighbors, how to maintain the information necessary to perform the routing. Apart from tuning the high level algorithm itself, many parameters such as timers and buffers have to be set.

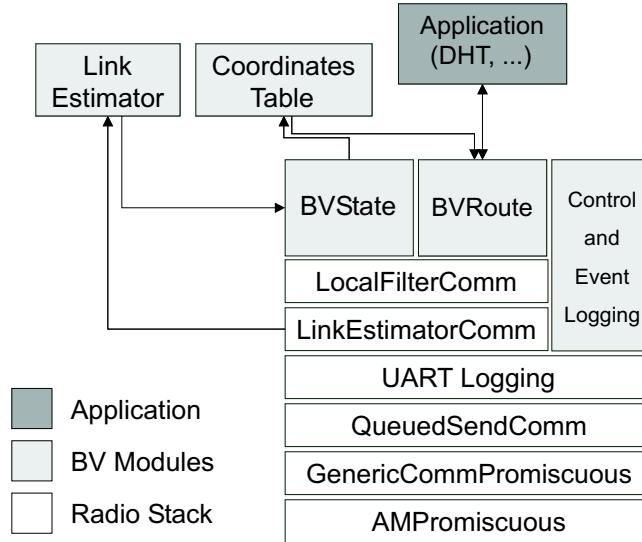


Figure 4: Block diagram representing the main modules of the implementation.

The fact that the same code that is to run on the real hardware is also used in the simulation makes it a really useful tool for debugging, as one can use `gdb`, for example, and also generate textual output to the console. This has advantages over debugging directly on the motes: using three leds for status information, and not being able to guarantee that all radio packets are received by a base station may introduce uncertainties in the debugging process. The scripting capabilities of the simulator are proving also very useful for testing specific and reproducible situations that the algorithm faces, in a controlled environment.

Programming in TinyOS is done using an event based model, and most but the more trivial operations are split-phase. We use the now standard networking abstraction in TinyOS to send and receive both radio and serial (UART) messages, namely Active Messages.

The main modules are shown in Figure 4, and we describe details of the implementation in what follows. Considerable amount of code is dedicated to logging, testing, debugging, and interacting with the motes. It is important to avoid the need to reprogram the motes frequently to perform small adjustments in parameters.

4.1 Networking Stack

The formal use of interfaces in TinyOS forces a separation of the specification of a functionality from its implementation, and it proves very beneficial in allowing very flexible and transparent compositions of functionality. By using the standard `SendMsg` and `ReceiveMsg` interfaces that implemented by the `GenericComm` module, we are able to compose a stack of filter modules that perform different functions on packets as they go through both the send and receive paths.

The white blocks in Figure 4 show the modules we use in the network stack. We use the 'promiscuous' variants of `AM` and `GenericComm`, to allow for the Link Estimator snooping all traffic that can potentially be received by the mote. On top of these two modules, moving up the stack, we have a send queue for packets, using the also standard `QueuedSend` module.

In our deployment, we use a network of motes which have an ethernet connection (via their serial ports) as a back channel for reliable interaction and data logging. This functionality is welcome when evaluating and debugging a multihop routing algorithm, which can become tricky when the very routing is the only way to

get to a certain mote. To make use of this channel, we interpose the `UARTLoggerComm` module in the network stack. This module duplicates to the UART port all packets that are sent and received by the mote. The receive events are processed right away upon receiving a message, while the send events are only processed after the `sendDone` of the bottom layer returns, and only if successful. This is to assure that every message logged as sent was really sent over the air. The UART logging subsystem is also responsible for sending special event logging packets that are generated by events other than radio messages.

The next module up is the `LinkEstimatorComm` module, described in the next subsection, which is responsible for feeding information from the packets seen to the Link Estimator.

The `FilterLocalComm` module is responsible for filtering the packets that are not destined for the local node. The result for the modules located above it in the stack is therefore the same as if the AM and `GenericComm` modules were not used in the promiscuous mode.

4.2 Link Estimation

Link estimation is a fundamental module of the implementation of multihop routing in sensor networks. The link estimator we use is based on the link estimator described in [14]. Link quality is estimated by assigning sequence numbers to all outgoing radio packets, and determining, upon the receiving of packets from other nodes, the number of packets received and the number of packets missed. The link estimator is allowed to snoop all network traffic it can, and we do this by using the AM layer in promiscuous mode. A node also sends out, periodically, the list of incoming links it is currently monitoring, with their respective qualities. This info allows the other nodes to learn the quality of the reverse links to them, which is very important for choosing good links along which to send routing messages. We perform link estimation by two components, `LinkEstimatorComm`, and `LinkEstimator`. The first is part of the network stack, and is responsible for three things:

1. Tag outgoing radio packets with a link-layer sequence number and the node it is being sent from
2. Read the sequence number and last hop from received packets, and communicate the receipt of the packet to the `LinkEstimator`.
3. Periodically send a `ReverseLinkEstimation` message

All outgoing packets are tagged, as they can offer valuable information for all neighboring nodes about the quality of the link from this node. When we say quality we are referring to the probability of a node receiving a message sent to it. The `ReverseLinkEstimation` message sends information about a subset of the links currently in the `LinkEstimator`, and the links to be included are determined in a round-robin fashion: if there are n links and the packet holds information about p links, each link will have its information sent out at least every $\lceil n/p \rceil$ periods.

The `LinkEstimator` does its estimation by maintaining a running average, an exponentially weighted moving average taken on successive windows of time. There is a periodic timer that triggers the updates to the quality of the incoming links. The number of received and missed packets in the last period are used to determine the fraction of messages sent that were received. This module is used by the rest of the information maintenance algorithms, and for the routing module as well.

The link estimator, as described, can only detect that a packet was missed when it receives a future packet, and a gap is noticed in the sequence numbers. In the case that a node ceases sending data for an extended period of time, the loss would be undetected. Our approach to this problem is as follows: when a window passed and we have 0 messages from a given node, we assume the quality in that window to be 0 also. There

is an underlying assumption here that the nodes periodically send data with a minimum frequency of at least one packet per link estimation window.

In order to expire a link, we count the number of time windows in which there were no packets received from the node, and expire the link when this number is less than the `AGE` threshold set.

The other issue that we discuss is that of replacement in the link table. In this prototype we did not implement replacement of links: a link shall only leave the table if it expires. For the prototype, we allow for space that it is enough for most of the time. According to [14], a policy that works well in practice is LFU, and we are going to test that more thoroughly as a next step.

4.3 Coordinate System Maintenance

We now describe the maintenance of the coordinate system information, required for BV routing. This functionality is provided by the `BVState` module in Figure 4. There are two key pieces of information that nodes must maintain:

1. The distance in hops to the root beacons (the node's own position)
2. The positions of the node's neighbors (at least one hop neighbors)

These are implemented in the prototype as two different flows of packets, largely independent. For maintenance of the root beacon distances, we have each root periodically send out flood messages to the network. The interval between two floods initiated by each root beacon is jittered, to avoid synchronization effects among the different floods. We guarantee that each node will send each message once, by having the root beacon identify each message with a sequence number and its *id*. Each node keeps a parent and a hopcount for each root beacon. Upon receiving a flood message originated at a root beacon, the node determines if it came from the current parent, or from a 'better' parent. In the latter case, the node substitutes the parent, and in both cases, it forwards the message. We define a 'better' parent as one that has a path to the root with the smallest combined expected packet loss rate along the path to the root. We get this information by combining the reverse path probability from the parent up to the root with our reverse link quality estimate for the parent itself. For this combination we use the product of the success probabilities along the path. One possibility for increasing the resolution for the lower end of the quality spectrum is to apply a logarithmic transform to the qualities, but we are not currently doing so.

The goal of such estimates is to avoid selecting long, low quality links that result in low hopcounts to the root, albeit of very low quality. We want to avoid the illusion of being close to the root when in fact any reasonable path would have a higher hopcount. It is true that in most cases we do not route towards the root, so a valid question to ask is why did we pick this metric as a good one. This is a sensible approach that yields good routes to the roots, and possibly selects a set of good links on which to route, too. One direction of future study is to compare this parent selection criterion to others, such as selecting lower hopcount links filtered by a quality threshold.

In the broadcast forwarding step, to minimize collisions when siblings are to forward messages from a parent, we enforce that nodes wait for a random delay before broadcasting. Also, we ensure that the nodes only forward messages that come from a parent, and that they ignore any messages that they have recently seen.

For the other part – maintaining the neighbor positions – nodes periodically send out their coordinates, which can be heard by the neighbors. Upon hearing the message from a neighbor with the neighbor's coordinates, these are stored in a table maintained by the `CoordinateTable` module. The requirement, though, is that the node the message was heard from must be in the table of links. These tables are kept in synch by having the `LinkEstimator` module signal when a change occurs in its table of links. We have provision in the

`CoordinateTable` to store coordinates of nodes which are not first hop neighbors, although this is currently not being used. To avoid synchronization effects, the interval between successive position broadcasts is also jittered around an expected interval value.

4.4 Routing

For routing packets, the `BVRoute` module (see Figure 4). The interface exported by this module is `route to position`, and is the basis with which we perform most of our evaluations. We leave it to the specific application to utilize this service, including implementing one of the suggestions in 3.3.

The algorithm we use for the routing is the same as described in Algorithm 1, and is implemented in the `BVRoute` module. When a message is to be forwarded, the `BVRoute` module requests that `BVState` provides a next hop, given the destination’s position, and whether the packet is in fallback mode or not. `BVState` then chooses a next hop based on the information preset at the `CoordinateTable`. Only the nodes in the `CoordinateTable` with an outgoing quality of more than 75% are considered active, and eligible for routing. We are currently testing the need for per hop retransmission.

4.5 Interaction and Logging

The last piece we describe is the `BVCommand` module, represented by the ‘Control and Event Logging’ box in Figure 4. It is responsible for receiving command packets, either through the radio or through the UART, and returning the responses. These commands include commands to get information about the state of the nodes (e.g., link table, coordinate table, current position and parents, radio power, compilation time/version), and to set different parameters on the mote (radio power, whether or not the nodes is to be a root beacon, and different timers). Other very important commands are a `routeTo` command, which tells the node to send an application message to a given position, and a `Freeze/Thaw` command, which instructs the mote to stop (and resume) updating all state (link, coordinate table, position information), thus enabling isolated testing of routing.

The logging of changes in the algorithm state is performed by a dedicated module that implements the `BVLogger` interface. This module, for the current mote implementation, generates AM packets that are handed to the `UARTLogger` and sent over the serial port. When running in the simulated version, on the PC, the logging module is substituted by one that just prints the contents of the packet as a debug message. The output produced is identical, however.

5 Evaluation

The evaluation of the performance of the implementation is done in several steps that build upon each other. Here we present results that evaluate the link estimator performance, the ‘coordinate system’ coherence and volatility, and some results on routing. In these results we do not explicitly test node failures, node additions, or mobility, which is the subject of continuing efforts. We do however test in a real radio environment, which presents some variation on the quality of the links over time, in parallel with simulations. We do not present here the result obtained in the high level simulator, which are being used for setting parameters on the algorithm itself, and testing the scalability of the algorithm to considerable larger networks of sensors.

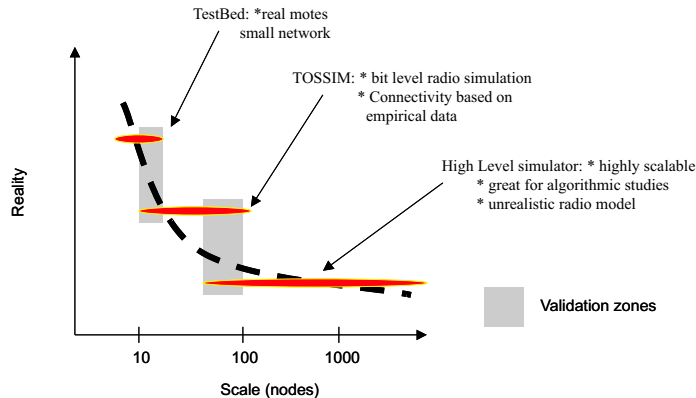


Figure 5: The framework for our evaluation of the algorithm, from a high level simulator to a real small deployment on a sensor network.

5.1 Methodology

In designing and evaluating the BV Routing algorithm, we find that there are many questions to be answered at very different levels. This is in part due to the development of the algorithm itself, but also largely due to the environment in which it is intended to run. The characteristics of wireless embedded networks impose new challenges, many of which are still under study by the research community.

To overcome this challenge, we adopted a multi-level approach, which can be described by Figure 5.1, inspired by a similar discussion in [1]. To develop high level parameters of the algorithm, such as the nature of the coordinate system, and how to find a suitable dissimilarity metric that bears good correlation with true network distance, we use a high level simulator, written in C++, with fairly simple assumptions on the radio connectivity of the nodes, namely a circular radio model with the assumption of no loss or congestion in the links.

In this simulator we have a very high turnaround time for results of variants of the algorithm in large scale networks of more than 12000 nodes. We also were able to explore issues such as the number of root beacons needed to maintain a certain performance, for different sizes and densities of the network.

Also at a very high level, we were able to gain intuition on the dissimilarity metrics, beacon placement, and number by studying a geometric analog of the algorithm in the euclidean space.

To account for a more realistic environment, we resort, on a level closer to reality, to a very detailed simulation environment, TOSSIM [4], and to a real deployment in a small experimental testbed at the Intel Research Laboratory in Berkeley. This consists of 16 nodes, connected by an ethernet backchannel for logging and control of experiments. These two environments run the same TinyOS code, which is very important for developing purposes.

The simulations in TOSSIM model the radio at the bit level, and are able to reproduce loss and contention for the channel². We are currently able to run TOSSIM simulations of up to 100 or 200 nodes, which enable us to cross validate results with the higher level simulator at these scales.

²In fact, the congestion model of TOSSIM is pessimistic, in that it does not take signal attenuation with distance into consideration. As such, if nodes can transmit to each other, even with a very high loss probability, they can still interfere substantially.

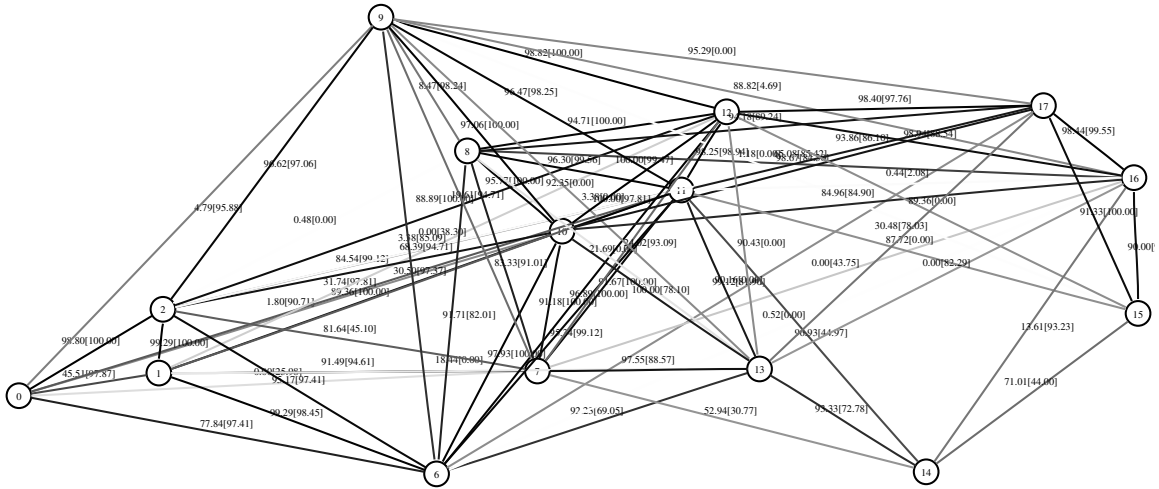


Figure 6: Graph of the link qualities measured in the testbed with 15 active nodes. The gray level indicates the average of the reverse and forward direction qualities. The numbers indicate the qualities, with the reverse direction in square brackets. For each link, forward is the direction from the lower to the higher numbered node.

5.2 Environment

The evaluations presented here are conducted in two settings. The first is a simulated network of 20 nodes in TOSSIM, with nodes randomly distributed, and a radio model derived from previous empirical measures. In this radio model, the link qualities are attributed to each link following a probabilistic model, as a function of their distance. There are four nodes acting as root beacons, and they are placed in four opposite corners of the network. In this environment there are no changes in the positions or link qualities over time.

The second environment, as we mentioned previously, is an experimental deployment of 16 Berkeley `mica2dot` nodes at the Intel Research Lab in Berkeley. These nodes use the Chipcon CC1000 radio chip, operating at a frequency of 433Mhz, and have an ethernet back channel that allows complete logging of their activity and state, as well as a convenient way of driving experiments. The environment is that of an office building, and there are several natural obstacles, interference, and tunneling effects that affect the radio connectivity. We do find, however, that the links are quite stable in quality, with some changes observed in the scale of 5 to 10 minutes. We also use 4 root beacons in this setting. In Figure 5.2 we show the graph of the network formed by the nodes in the testbed, with the respective link qualities among nodes. We observe a similar behavior in the correlation between distance and reception probability as what is related in [14].

In Table 2, we list the main parameter settings that were in use for the experiments. In order for the protocol traffic not to interfere significantly with the routing traffic, we set the periodicity of the messages sent rather long. We can derive some numbers from the table. For example, the expected number of messages sent by a node per second is on the order of 1 every 10 seconds³. Given that the capacity of the radio is around 30 messages per second, at this rate we can easily accommodate 30 nodes in range of each other, occupying around 10% of the channel.

³There are 4 root beacons, which means the node is expected to forward 1 root message every $(60/4)$ seconds. The node will send an expected 1 position broadcast message per 40s, and an expected 1 reverse link info message every 240s. The expected combined rate is, then $\frac{23}{240}$ messages per second.

Link Estimator	
Size of Table	12
Expiration	5 successive windows with quality \geq 15%
Replacement	Only by expiration
Reverse Link Info Period	240s (after 3 periods of 80s \pm 40s jittered)
Update Link Time Window	120s (fixed)
Exponential Average	History of 25%
BVState	
Root Beacon Timer	60s \pm 30s (uniform)
Position Broadcast Timer	40s \pm 20s (uniform)

Table 2: Parameters used in the evaluations of the implementation, for both the TOSSIM and testbed runs

5.3 Preliminary Results

Before evaluating the routing performance itself, we must evaluate whether the link estimation produces a consistent picture of the network and is able to correctly identify links that are good and links that are bad. Figure 5.3 shows the output of the link estimator running on the motes on the testbed, versus the link quality measured over the same period of time, by collecting the actual traces of packets sent and received. This data is collected from the testbed as mentioned above, and corresponds to 30 minutes, with just the protocol maintenance traffic. The parameters used are those in Table 2.

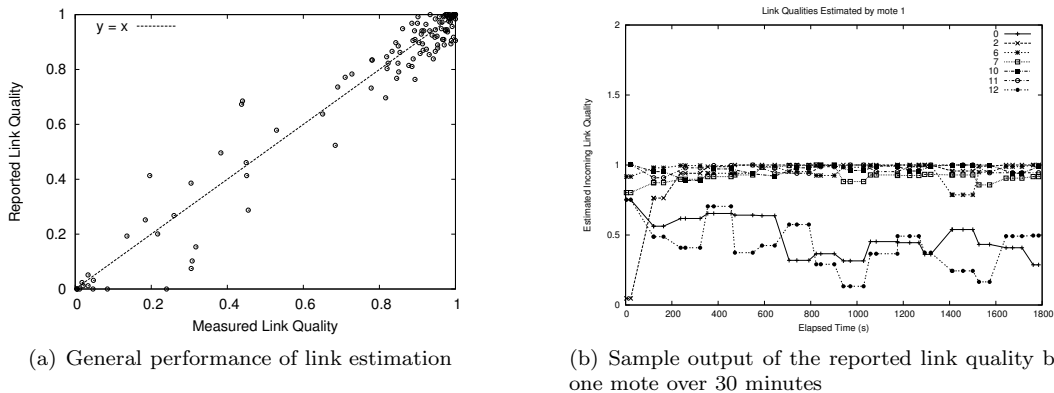


Figure 7: Performance of the link estimation in the real deployment. Measured link quality is the fraction of packets sent by the originating mote that were received by the destination mote, as determined from the log generated by the motes. Reported link quality is the quality reported by the link estimator running at the destination mote

Our findings with the link estimation is that it produces good results if the actual link quality is above 75%, and presents some variation for links of intermediary quality. We show some example curves in Figure 5.3(b). In this Figure we plot the successive estimations reported by a mote over the course of an experiment. This is acceptable, since we only use for routing links which are above this threshold, but we also believe that the estimator can be enhanced. There is a tension between the agility of the EWMA estimator and the error that it produces. From the results in [14], we are investigating giving more weight to the history. Another possibility is to take the confidence interval of previous values into consideration when computing the new estimate, given a new observation [13]. This is a part of the evaluation that cannot be performed in the simulation, but only in the real deployments.

We also have some results on the stability of the network over time for the Testbed, given the natural variations observed in the link quality. We intend to perform more experiments, explicitly shutting down nodes, to observe the resulting changes in the coordinates and the healing time of the network. These tests can be done both in simulation and in the testbed. Figure 5.3(a) depicts the tree for one of the root beacons (node X), in an experiment on the testbed after x seconds. Figure 5.3(b) shows the evolution of the tree over time. There is one curve for each node, plotting the node's depth in the tree versus time. Although it is hard to see ⁴ in this plot, we can notice that the transitions aren't too frequent, and that they usually differ by one hop only. We are interested in looking at the transition data in the presence of node failures and mobility.

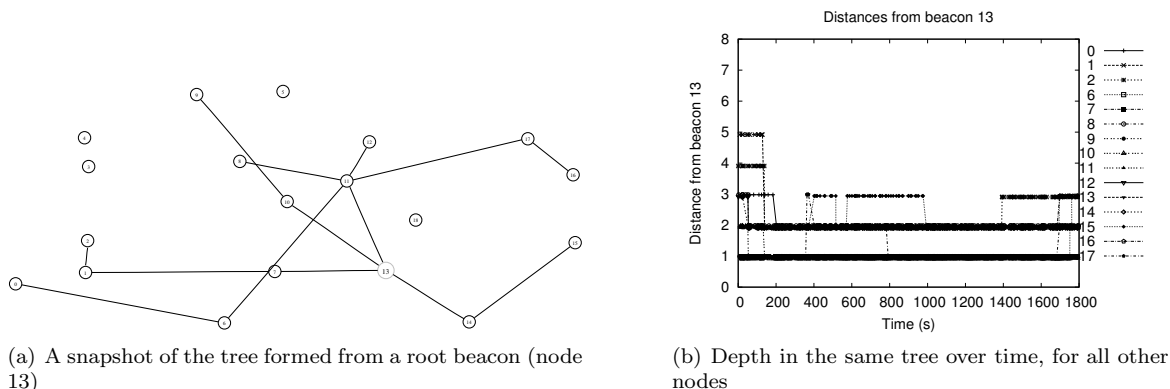


Figure 8: The tree formed for one of the 4 root beacons in our experiment, and how it evolved over time. Note that after an initial settling time, the transitions become infrequent and with a difference of 1 hop in most cases.

We are in the process of executing systematic tests on the performance of the algorithm, with the goal of conducting similar automated tests on the simulator and on the testbed. So far we have very preliminary results of the routing performance, which we list on Table 5.3. The experiment consisted in these cases of a set up time of 30 minutes (of real or simulated time), for the coordinate system, neighborhoods and link estimations to be established, and a set of routes initiated from the nodes with given destination. After the setup time the routing state was 'frozen', and the protocol traffic to maintain it ceased. The goal with this rather artificial setup was to evaluate how well the algorithm is choosing the links to route on. In both environments, we executed routes from all nodes to all nodes, in a round robin fashion. There was only one message being routed at any time in the network. To obtain the coordinates of the destination, we queried it using the commands module right before issuing the `route` to command from another node.

These results are not directly comparable to each other, as the characteristics of the radios in the two environments are different. We intend to try to approximate the radio characteristics of the testbed in the simulation in order to validate its results for larger scale simulations. As we can see, between 13% and 10% of the routes required scoped flood, but for these two cases, all of these only required a flood of one hop, i.e., equivalent to one more hop in the route. Of course, these are very small networks, with diameter of 3 or 4 hops, and we need results for larger networks. We are still compiling results for route length and number of fallback steps.

Regarding benchmarking the algorithm as to the quality of the routes, we intend to compare the routes obtained with the optimum route in the graph induced by the neighborhood tables of the nodes, optimizing either for hopcount or for probability of success. This information can be processed offline from the state of the routing tables at a given moment.

⁴I am thinking of better ways to convey information about these positions

Route	Testbed			TOSSIM		
	Total	%	% C	Total	%	% C
Started	408			740		
Dropped	29	7.1%	-	235	31.8	-
Successful	343	84.1%	90.5%	436	58.9	86.3
Same Coords	0			0		
At beacon	36	8.8%	9.5%	69	9.3	13.7

Table 3: Preliminary results on routing for the two environments. We do not use any type of retransmission for these results. ‘At beacon’ stands for the routes that reached a root beacon in fallback mode, thus requiring the scoped flood to complete. We did not perform the scoped flood in these experiments. %C is the percentage of results over the routes that completed, i.e., did not have drops.

6 Conclusion and Next Steps

We plan to investigate alternative ways of maintaining the coordinate system, comparing against the current implementation in terms of traffic generated and energy consumption of the network. One such possibility is the unification of both types of maintenance traffic mentioned in Section 4.3 – the beacon floods and the position advertisement – into a single advertisement mechanism, similar to the implementation of the traditional Distance Vector routing protocol. In this scheme, nodes derive their distance from the root beacons from their neighbors positions. The advantages of such a scheme are the simplification of the protocol, and the absence of time-correlated transmissions. Disadvantages are the possibility of the appearance of count-to-infinity problems upon certain disconnection situations, and the slower propagation of information in the network. However, it seems to be a promising direction.

Another interesting aspect to look at is that of adjusting the rate of the information dissemination depending on the stability of such information: if the positions are not changing, the rate with which a node advertises its information decreases.

There is much work yet to be done. In terms of evaluation, we must look at the evaluation of

- different schemes to choose ‘parent’ for a given root beacon
- effect of node and link failures (regular nodes and root beacon nodes)
- effect of node and link additions (specially the appearance of long links with good quality)
- scalability (larger and denser networks)
- mobility
- root beacon placement
- performance of different lookup mechanisms (DHT, consistent hashing to root beacons)

In particular, we want to more carefully study the traffic generated and scope of changes in links and in the availability of nodes, and how the coordinate system is affected by such changes. This has important impacts for example if one is storing data in the network which is addressed by a particular key, and thus, a particular position.

Initial evaluation has shown that the algorithm is able to produce efficient routes, close to the shortest path routes, and that it is possible to reduce the need for routing in the scoped flooding mode to less than 10% of the routes. The algorithm is simple, and the high level simulations indicate that it is scalable. The fact

that the routing gradient is derived solely from the connectivity information makes than algorithm perform better than geographic routing in the face of obstacles, network ‘voids’, or lakes, and at low network density.

7 Acknowledgments

This is joint work with Sylvia Ratnasamy, from Intel Research, Berkeley; Ion Stoica, and Scott Shenker, from UC Berkeley; David Culler, from UC Berkeley and Intel Research, Berkeley; and Jiong Sheng, from UC Berkeley. I would like to thank the TinyOS team at Intel Research for all the help and support, specially Philip Levis, Alec Woo, and Cory Sharp. Also, I would like to thank Benjamin Greenstein for many conversations about gritty details of the implementation.

References

- [1] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. Emstar: An environment for developing wireless embedded systems software. Technical report, CENS Technical Report 0009, March 2003.
- [2] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems, ASPLOS’00*, pages 93–104. ACM Press, 2000.
- [3] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM Press, 2000.
- [4] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 126–137. ACM Press, 2003.
- [5] James Newsome and Dawn Song. Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 76–88. ACM Press, 2003.
- [6] T.S.E. Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, volume 1, pages 170–179. IEEE, 2002.
- [7] Ananth Rao, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108. ACM Press, 2003.
- [8] Sylvia Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin, and Fang Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mob. Netw. Appl.*, 8(4):427–442, 2003.
- [9] E. M. Royer and C-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 6:46–55, April 1999.
- [10] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.

- [11] Liying Tang and Mark Crovella. Virtual landmarks for the internet. In *Proceedings of the conference on Internet measurement conference*, pages 143–152. ACM Press, 2003.
- [12] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *Symposium proceedings on Communications architectures and protocols*, pages 35–42. ACM Press, 1988.
- [13] Alec Woo. Personal communication.
- [14] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 14–27. ACM Press, 2003.
- [15] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 1–13. ACM Press, 2003.