GRAViTy: Geographic Routing Around Voids in Sensor Networks

Tassos Dimitriou
Athens Information Technology, 19.5 km Markopoulo Ave., 19002 Peania, Athens, Greece
Email: tdim@ait.edu.gr
Ioannis Krontiris
Athens Information Technology, 19.5 km Markopoulo Ave., 19002 Peania, Athens, Greece
Email: ikro@ait.edu.gr

Received: January XX 2005; revised: November XX 2005

***Abstract*— Nodes in sensor networks do not have enough topology information to make efficient routing decisions. To relay messages through intermediate sensors, geographic routing has been proposed as such a solution. Its greedy nature, however, makes routing inefficient especially in the presence of topology voids or holes. In this paper we present GRAViTy (Geographic Routing Around Voids In any TopologY of sensor networks), a simple greedy forwarding algorithm that combines compass routing along with a mechanism that allows packets to explore the area around voids and bypass them without significant communication overhead. Using extended simulation results we show that our mechanism outperforms the right-hand rule for bypassing voids and that the resulting paths found well approximate the corresponding shortest paths. GRAViTy uses a *cross-layered* approach to improve routing paths for subsequent packets based on experience gained by former routing decisions. Furthermore, our protocol responds to topology changes, i.e. failure of nodes, and efficiently adjusts routing paths towards the destination.**

***Index Terms*— Geographic routing, Topology, Voids, Greedy forwarding, Wireless networks**

I. INTRODUCTION

During the past few years there has been an explosive growth in research devoted to the field of wireless sensor networks, covering a broad range of areas, from understanding theoretical issues to technological advances that made the realization of such networks possible. Routing has become the foremost problem in such networks. Due to the energy constraints of sensor nodes, routing involves relaying messages through a series of intermediate nodes from source to destination. Moreover, the memory constraints and communication overhead involved do not allow the use of routing tables as in wired networks. So, in random topologies the network has to discover routes that fulfill certain criteria such as minimum power utilization and/or minimum path length.

One of the proposed techniques for routing in sensor networks is geographic routing [1], [2], where each node has knowledge of its position as well as the position of the base station, and therefore can forward the data packets closer to the destination. Geographic routing is efficient in dense networks where the packet can always be forwarded under this greedy forwarding strategy. However, in more sparse topologies greedy forwarding may fail to find a path towards the destination even though such paths may exist. In these cases the packet reaches an intermediate node that has no neighbors closer to the destination, so making a greedy choice cannot result in any further progress. Therefore, alternative strategies must be tried until greedy forwarding can be used again.

To overcome these local minima and help packets advance further in the network, [3], [4] propose the use of the "right-hand rule" that routes packets counter-clockwise along a face of the graph until they reach a node that is closer to the destination than the one where the packet entered this perimeter traversal mode. However, as we will see later, this solution does not provide efficient routes for voids that do not have a closed (convex) shape. Furthermore, it requires the extra cost of graph planarization which eliminates several edges of the graph. This usually results in longer paths as the node has less choices for forwarding a packet.

In this work, we propose a simple mechanism to help packets overcome local minima. When the packet cannot be forwarded to a node closer to the destination, still a greedy choice is made from the other neighbors of the node, even if that means the packet will head backwards. By making sure that the packet is not sent twice to the same node, we eventually reach a node where positive progress can be made. While early packets make this additional effort of "discovering" the topology, we employ a *cross-layered* approach to take advantage of the experience gained by this effort and improve energy and communication efficiency of routing subsequent packets. We create an interdependency between the physical and the routing layer to relate routing decisions of nodes with those of their neighbors and improve the routing paths.

GRAViTy (Geographic Routing Around Voids In any TopologY of sensor networks) is a localized routing protocol which efficiently produce paths that compete with the shortest paths under the presence of topological voids. The protocol has the following properties:

1) *Direction-based routing*. Each node estimates the direction of the base station as well as that of its neighbors and forwards the packet to the node with the direction closest to the *direction* of the base station. The paths produced are single-paths.
2) *Localization*. Each node makes decisions based solely on local information, that is information gained from facts within its neighborhood. This includes the location of its neighbors with respect to the base station as well as routing decisions that they make.
3) *Loop-freedom and memorization*: Under the presence of routing holes, there may not always exist next hops with positive advantage towards the destination. In this case, localized, greedy algorithms are not loop-free unless they use some kind of memorization. Some information about past traffic must be stored either in the routed packet or in the nodes. Keeping this extra information in the packet increases its length, and makes transmission by nodes more expensive. Our protocol stores some information about past traffic only in certain nodes and only for a short period of time. Moreover, the information stored is bounded, since it concerns traffic only in the neighborhood of each node.
4) *Minimization of distance traveled*. Our protocol optimizes the distance traversed by the routed packet, using only local information. It turns out that the resulting path is very close to the shortest path from source to destination.
5) *Scalability*. Our routing algorithm performs well for an arbitrary number of nodes. Scalability is tightly related to the notion of localization. As long as each node selects the next hop based solely on local information, the performance of the algorithm is not affected by the network size.
6) *Guaranteed message delivery*. Our algorithm guarantees message delivery provided the network remains connected.
7) *Robustness*. The accuracy of destination of the base station and that of neighboring nodes does not affect the efficiency of our protocol.

In what follows we assume that the number of sensor nodes in the network is N and there is a single destination point D that represents the center where data should be sent. We denote the node that sensed the event by S. We assume that each node has the following capabilities (in Section IV we will see how these assumptions are validated by current technological advancements):

1) It can estimate the Direction of Arrival (DoA) of a received transmission, and
2) It knows the general direction of the base station D.

We also assume that the sensor nodes are statically located after deployment. Hence we do not consider here a dynamic sensor network, where sensors are mobile. Finally, note that each sensor node is *not* assumed to know its location, since our forwarding strategy is based on angles (direction) and not on coordinates (position).

The rest of the paper is organized as follows: In Section II, we discuss related work and in Section III we describe GRAViTy in detail . In particular, we start by showing how our strategy compares with perimeter routing under the presence of different types of voids and then show how to enhance our protocol by looking at subsequent packets that take advantage of the knowledge gained in the past. We also demonstrate how the protocol adjusts to topological changes due to node failures. In Section IV, we discuss some implementation issues while in Section V, we present our experimental results and argue about the efficiency of our protocol. Finally, we conclude in Section VI.

II. Related work

In early work on geographic routing [1], [2] the notion of greedy forwarding was introduced, where the location of a node is taken into account in order to make progress towards the destination. However, greedy forwarding fails when a node has no neighbors closer to the destination. A similar scheme has been proposed in [5] where the direction of the neighboring nodes is used as the criterion for greedy forwarding.

Under the presence of routing holes, greedy forwarding fails and alternative strategies must be used in order to make progress until greedy forwarding can resume. As we already mentioned, one popular solution is *face routing* [3], [4], [6] (also called perimeter routing or planar graph traversal), which uses face changes and the right-hand-rule to route around the void. In order for face routing to work correctly, the nodes must run a distributed algorithm that planarizes the network graph. Besides the extra overhead this operation imposes, it eliminates edges from the graph, resulting in less possible choices for nodes to forward packets and therefore inefficient path lengths.

A proposal to replace the right-hand rule by distance upgrading is presented in [7]. During an initial phase each node learns its distance to the base station. The packet is always forwarded to the neighbor with the smallest distance. The authors propose a way to transform the routing graph by artificially increasing the distance value of dead-ends, so the packet is never forwarded to them. Their strategy however requires an additional overhead of control packets until all dead-ends are removed by the network. This overhead is proportional to the number of voids and the network size, and can be increased substantially for sparse networks. Furthermore, without the presence of artificially created holes, the algorithm behaves the same as GPSR in random networks.

Other strategies have been proposed for bypassing routing holes that also avoid the use of perimeter mode. In [8], the base station is reached by having nodes memorizing the shapes of holes so that when a packet gets stuck the algorithm computes the shorter side of a hole and forwards the packet accordingly. However, when holes are large, the high communication overhead and memorization in nodes along the holes is increased.

The use of depth first search for route discovery in geographic routing has been proposed in [9] and [10]. In [9], each node puts its name and address on the packet and forwards it

to the neighbor that minimizes the Euclidean distance. Those neighbors that have forwarded the packet in the past are excluded from the legitimate candidates. However, this kind of information in the packet increases its transmission energy and makes the protocol less scalable for large network sizes. In [10], the authors show how to use DFS in order to construct QoS paths. The whole DFS path from source to destination is followed, assuming the use of GPS. The nodes on the created path memorize both the previous and the next node on the path. Each time a node receives the same packet twice, it returns it to the sender in order to avoid loops, resulting in excess transmissions. Furthermore, no power consumption model is assumed, so the energy efficiency of the algorithm is not shown, and no comparison with GPSR is attempted.

A completely different forwarding strategy in geographic routing is the restricted directional flooding. For example in [11] a protocol is presented where information on a sensed event is propagated towards a receiving center by activating only those nodes that lie very close to the optimal path between the source of the event and the destination. By changing a parameter of the protocol, the average size of the propagating front of nodes can be configured, where the front is simply the nodes that lie at the edge of the transmission zone towards the destination. If this front is set bigger than an obstacle or void, then obstacles can be bypassed.

Finally, the use of a cross-layered approach has been proposed in [12], where it has been pointed out that truly efficient use of network resources and optimization of end-to-end quality in wireless networks requires exchange of information across the layers that would not be possible with the traditional layer interfaces. Recently, the cross-layered approach has been used to geographical routing in sensor networks in order to improve its energy efficiency [13].

III. GRAViTy

In this section we discuss GRAViTy in detail. We break down the description in various subsections and "rules" to ease the readability of the protocol and motivate the need for each "enhancement".

A. Routing a single packet

We start by describing how nodes can forward a single packet, assuming no prior routing history of the network. When a node, say F (Figure 1), receives a packet for destination D, it needs to decide which of its neighbors the packet should be forwarded to. Let $\phi_i = (FP_iD)$ be the angle between node F, its neighbor P_i and the destination D. Then F forwards the packet according to the following local rule:

Rule 1: Each node F forwards a packet to the neighbor P_i with the maximum angle $\phi_i = (FP_iD)$.

When node P_i receives the packet from node F, it will mark node F as its *parent* and F will mark node P_i as its *child*. In order for node F to decide which of its 1-hop neighbors has the best forwarding angle, we assume that it keeps in memory a table $(\langle P_i, \phi_i \rangle)$ with all its neighbors P_i and their corresponding angles ϕ_i. This table could have been created, for example, in an initial phase after deployment of the network where each node broadcasts a request to all its neighbors to send them the angle at which they would forward a packet originating from itself.

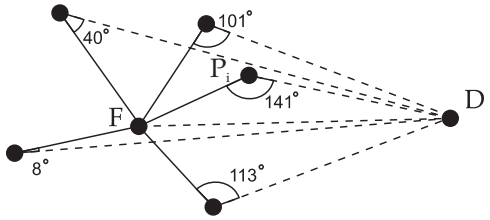

Fig. 1. Forwarding strategy based on direction. Node F forwards the packet to the neighbor P_i with the largest angle. In this example, the node with the 141° angle will be selected.

Neighbors compute this angle by using information available to them, i.e. the DoA of the received "request" signal and the direction of the base station, D. The table of angles created by the requesting node is not to be used as a routing table. It just stores information that will facilitate the application of Rule 1, and the rules to follow.

Angle ϕ can range from 0 to 180 degrees. The closer a node's angle is to 180°, the closer that node is to the forward direction towards the destination. A node with a small angle is a backward node. If there exists no node in the forward direction, we don't consider the node a dead-end. The algorithm will choose a backward node. Therefore, a packet following this greedy strategy does not always move closer to the destination.

The fact that the packet can go backwards means that eventually the packet may reach a node that is already part of the routing path, creating a loop. In this case we would have to drop the packet. But since we want to guarantee delivery to the destination, we need an additional rule to prevent loops.

Rule 2: A node cannot forward the packet to a neighbor, which has forwarded it before.

This suggests that a node F will choose to forward the packet to a neighbor P_i according to Rule 1, unless that neighbor has already forwarded that packet (belongs to the routing path), in which case node F will choose to forward the packet to the neighbor with the next largest angle. In order to realize that, we need to find a way so that by the time a packet reaches a node, that node knows which of its neighbors have forwarded the packet before and exclude them from the forwarding decision. If this is not the case, those neighbors will have to receive the packet and send it back to the sender node, resulting in two excess broadcasts. To avoid these excess transmissions we employ a *cross-layered* approach [12].

Wireless networks normally use a single-frequency communication model. When a packet is broadcasted, it is heard by all nodes in the transmission range of the sender. These nodes have to open the header of the packet (in the MAC layer), where the sender and recipient are included, and check if they are the indented receivers. If not, they stop receiving and

the information of the packet's sender and recipient addresses can be passed to the routing level of the node and be stored. So, when that node needs to make a forwarding decision, it will already know which of its neighbors have received and forwarded the packet, since that information can be found in the overheard packets.

Therefore, in order to prevent loops, each node that participates in the routing procedure must memorize traffic information in its neighborhood for a short period of time. This means that no extra information is stored inside the packet, which otherwise would increase transmission energy. The only memory requirement is that nodes must store the routing information of packets that were forwarded in their neighborhood in the recent past. As we will see in Section V, depending on the network size, only a few couples of integers need to be stored in each node, which is feasible for the memory space available in sensor nodes.

So, by receiving a packet, a node has all necessary information to select one of its neighbors to forward the packet to. However, there is the case where the packet has reached a local minimum and the only available neighbor is the node that sent it (the parent). Then we say that the packet has reached a *dead-end* and according to rules 1 and 2, it cannot make any more progress. The only way to recover from this local maximum is to send the packet back to the parent. We call this action *backtracking* and we modify Rule 2 as follows:

Rule 2 (revised): A node u cannot forward the packet to a neighbor that has forwarded it before. If no other neighbors exist, the packet is forwarded to the node's parent (backtracking). Then we say that node u is a dead-end.

When this is the case, the parent excludes u from future transmissions and eliminates it from its list of valid neighbors. In case other neighbors exist, the parent will pick the best one according to rules 1 and 2, and send the packet to it, updating its child pointer. Otherwise, it will backtrack to its parent and proceed accordingly.

Figure 2 illustrates the case of a dead-end. Node F has forwarded the packet to node P, which in its turn chooses to forward it to node B, as the best option. Node B has no other neighbors than P, so backtracking is necessary. As a result, node B sends the packet back to node P. Node P has only one option now: to forward the packet to node N, since F is its parent and B a backtracking node.

B. Bypassing topological voids

Often, in sensor networks we have to deal with "holes", where a node has no forward neighbors towards the destination. These holes can be formed either due to topological voids or by failure of sensor nodes due to a number of reasons (battery depletion, physical damage, malfunctioning). So, a routing path based on greedy forwarding may be blocked from moving closer to the base station due to the lack of relaying nodes to cross the void. In this case the packet must find its way by moving "around" the void.

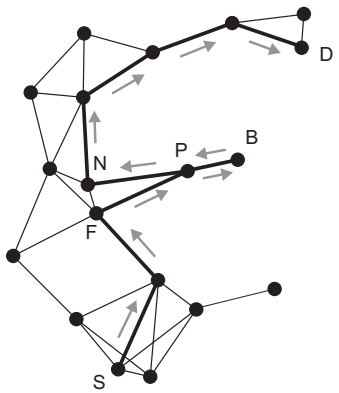

Fig. 2. Packet is routed from source node S to destination D. When the packet reaches a dead-end (node B) it backtracks and follows the next best path.

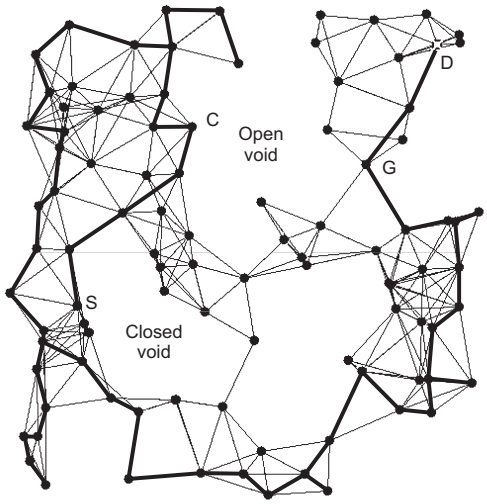

Fig. 3. Behavior of GPSR in the presence of voids. At node C, the packet enters perimeter mode exploring the perimeter of the graph and returns to greedy mode again at node G.

We distinguish between two different kinds of voids: closed voids, and open voids (see Figure 3). In closed voids, once a packet has reached a point at the face of the void where it cannot move further there are two different directions that it can travel in order to bypass it, even though one may be more efficient (i.e. shorter) than the other. In open voids, there is only one correct direction. The other direction will not lead to the destination, and therefore we need to head backwards and choose a different way.

By having only local information available, the routing path will have to "explore" the topology in order to find its way to the destination. Using the right-hand rule, inefficient paths may be produced, exactly because the counter-clock-wise direction is always used. This may cause the packet to be routed along the boundary of the whole graph, before it reaches the destination. This case is shown in Figure 3, where GPSR was employed. The packet starts at node S and is forwarded in greedy mode until node C. In node C the algorithm turns to perimeter routing and traverses the boundary of the graph until node G is reached (the first node closer to destination than C), where it turns back to greedy mode again.

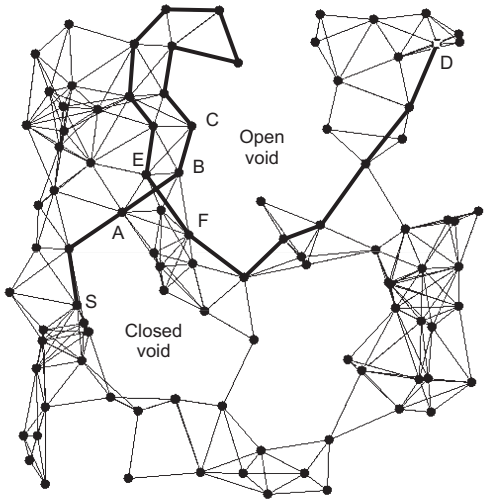

Fig. 4. Routing around voids with GRAViTy. The packet explores the topology around the open void until it finds the way through node F.

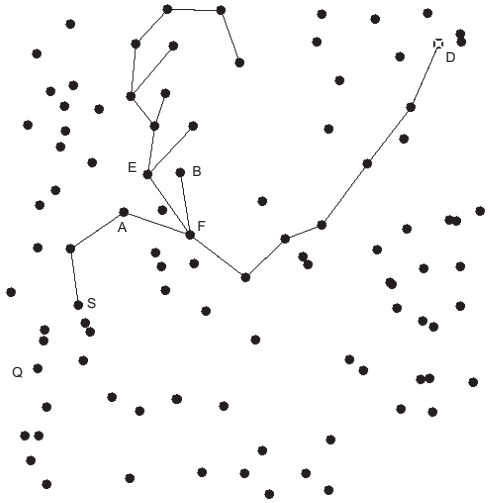

Fig. 5. Child pointers created, after a single packet has been routed from source S to destination D (as shown in Figure 4). Pointers are directional (not shown here) and point to the direction that leads to D.

In Figure 4 it is shown how GRAViTy manages to find a path to the destination. The packet (which is routed as indicated by the links in bold) reaches node C and cannot make any further progress towards the destination. By avoiding nodes that has visited before, the algorithm continues choosing the best possible direction according to the greedy criterion. However the packet does not always make a positive progress. It rather "exhausts" the area around the void, until it finds a way to move forward. Of course, it still possible that the packet may do a lot of unnecessary work, but as we will see in the experimental section this is rarely the case.

C. Routing subsequent traffic

In the previous sections we saw how packets have to explore the graph in order to find their way to the destination. Subsequent traffic could gain from this experience and save the effort of excess hops. We achieve this by storing a small amount of information in the nodes. In particular, the child pointer is the only information that is needed. When forwarding a packet through some neighborhood for the first time, the child pointers that are created in nodes can help improve substantially the routing paths of subsequent packets. We begin by defining the following rule:

Rule 3: When an intermediate node with its child pointer not being null receives a packet, it will forward the packet to the preselected child without making any other routing decisions.

So, a packet that is routed to the base station needs only to discover a path until it reaches a node that has a child pointer. Thereafter, it will follow a predetermined path, without any excess effort. This follows common intuition since if a previous packet has explored the topology to find an efficient path then any subsequent traffic that reaches the same node will have to follow the same path, eventually. So, we can gain from past "experience". However, there are three ways to improve upon this situation:

1) *Eliminate dead-ends*: When a node backtracks to its parent, then the parent can mark this child in its table and never forward a packet to it again. If the packet didn't find a way to the destination through that node, subsequent traffic won't find one either.
2) *Eliminate triangles*: When a node forwards the packet to one of its neighbor, and that neighbor forwards it to another neighbor of the initial sender, then a shortcut can be created bypassing the intermediate neighbor.
3) *Eliminate crossings*: When a packet is being forwarded across the neighborhood of a node *after* that node has forwarded the same packet, then the node can update its child pointer by overhearing to this transmission.

The latter case can be observed in Figure 4. Node E forwards the packet to node F. Node A that has forwarded that packet to node B in the past, upon overhearing this transmission can update its child pointer to point at its neighbor F. The next time a packet arrives at node A, it will be forwarded directly to node F, eliminating the "closed circuit" and saving 11 hops from the routing path. Note that in this case, the resulting path is also the shortest path. Moreover, the same broadcast will be heard by B as well, which since it has participated in the routing path it will also update its child pointer to F.

So, nodes that make a greedy choice and forward the packet can gain substantial information by the routing decisions of their neighbors. We formulate the above three cases as follows:

Rule 4: For a broadcast of a packet from node u to node v, each node that overhears it and has forwarded that packet before updates its child pointer to v, if v belongs to its neighbors, or else it updates it to u.

This applies also in the case of backtracking. Since the packet follows a path that does not contain loops (due to Rule 2), then updating child pointers according to Rule 3 will not create loops either.

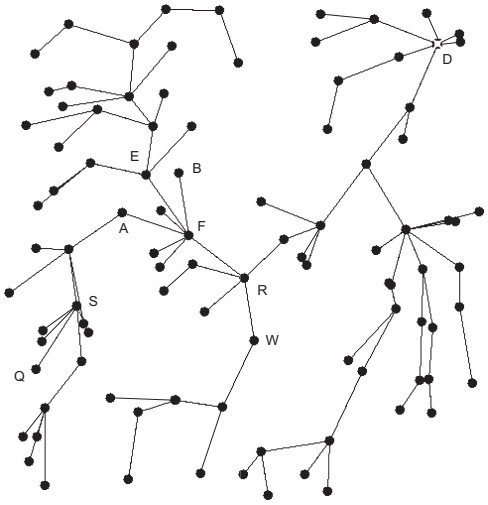

Fig. 6. Child pointers created after routing packets generated by random source nodes. They all point to the direction that leads to destination D.

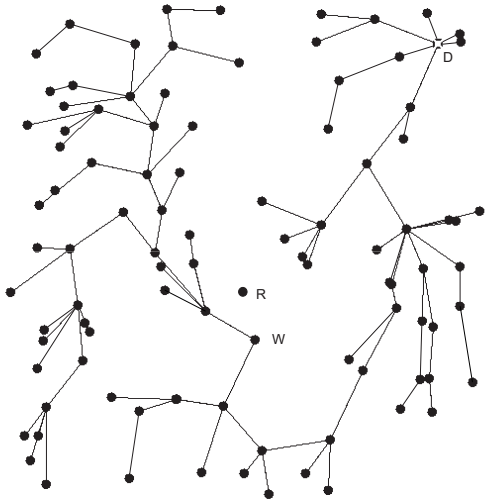

Fig. 7. Adjusting to topological changes due to node R failure.

The child pointers created from the routing of the packet from source node S to destination D, as we described in Figure 4, are shown in Figure 5. As we see, the excess communication effort of initial packets trying to reach the destination is not wasted. If a packet in an "unexplored" part of the graph takes a way that does not lead to the destination (because of its myopic strategy) and then have to turn back and choose a different way, it has still created the right child pointers on its passage to be used by other nodes that need to send data to the destination. So, if now node B needs to send data to D, it will directly forward the packet to node F, according to Rule 3, bypassing the greedy forwarding procedure that would result in excess hops. Likewise, node Q that does not have a child pointer would forward the packet to node S according to the greedy criterion, and from then on, the constructed path of child pointers would be followed.

We applied these rules to the network of Figure 4 by choosing nodes at random and have them generate packets which are routed to node D. The result is shown in Figure 6. In the figure, only the child pointers are shown and not the communication links. What has been created is a tree of paths that connects each node of the network with the destination through a single path. As we will see in the experimental section these paths are on the average about 7% longer than their corresponding shortest paths and a lot better than the paths produced by GPSR.

D. Dealing with node failures

Since our routing protocol creates single paths, we have to deal with node failures. A node may have its energy exhausted or fail unexpectedly, cutting-off paths that go through it. Then, these paths must be restored, bypassing the dead node.

Since nodes are blind beyond their neighborhood, it is the packet that has to re-discover a new path as if it is forwarded for the first time (Section III-A). A dead node may have created a new void or extended an old one, resulting in a completely different topology of the network, and thus the paths may have to change substantially. So, we define the following rule to deal with node failures:

Rule 5: When a packet reaches a node that its child has failed, then that node sets a flag in the packet indicating that any node which receives it should forward it by applying the greedy criterion all over again, ignoring its child pointer (if any).

That is, all the described rules so far still apply, except Rule 4. The node that its child node has failed will have to erase that node from its table and forward the packer again, applying Rule 1 and setting the flag. The same applies for the rest of the nodes: any node receiving that packet will erase its child pointer and decide where to forward the packer according to the greedy criterion, creating a new child pointer. The packet will be routed like it is the first packet in the network (as described in Section III-A) until it reaches the destination D.

For example, let's assume that node R (Figure 6) fails. Suppose that a new packet is again generated at node S and been routed following the discovered path until node F. Then node F will realize that it's child node R has failed. Node F sets the flag in the packet and forwards it to the node with the maximum angle (excluding of course node R). After that, the packet will be re-routed like it was the first packet in the network, creating new paths (i.e. child pointers), as shown in Figure 7. As it was expected, the next efficient path to reach the destination is through node W.

To show that the topology (and the routing paths) may have to change substantially due to node failures, assume that node W fails too. The resulting paths are shown in Figure 8. Comparing with Figure 7 we see that packets originated from the left part of the network now have to follow completely different paths. In Section V, we present simulation results that show how much this procedure of re-discovering paths burdens the efficiency of the protocol.

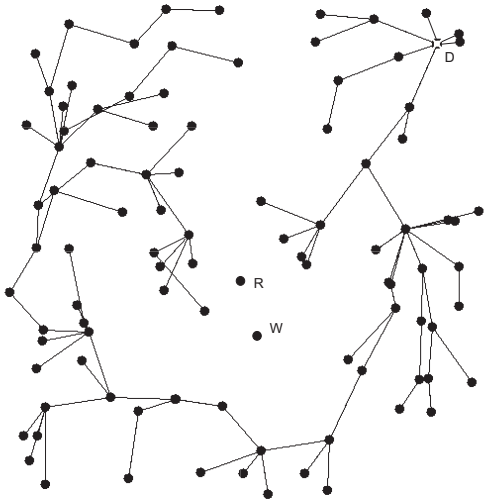

Fig. 8. Adjusting to topological changes due to node W failure.

```
if packet received {
 if child != null
   if sender == child {
     mark child as dead-end
     child = null
   }
   else
     forward the packet to child
 else if child == null
   if parent == null
     parent = sender of the packet
   if all of w's neighbors have
      forwarded that packet
     send packet to w's parent
   else {
     for all neighbors that have not forwarded
         the packet && are not dead-ends
       find neighbor N with the largest angle
     forward packet to N
     child = N
   }
}

if overheard packet transmission
   from node u to node v
  if w have forwarded that packet before
    if v belongs to your neighbors
      child = v
    else
      child = u
  else
    store "u has forwarded packet p"
```

Fig. 9. Algorithmic description of the GRAViTy protocol for each node w of the network.

E. Summarizing the protocol

Figure 9 summarizes the GRAViTy protocol for routing a packet according to the rules we have presented so far. Any data structures needed to store the necessary information for this algorithm should result easily from the algorithmic description. However, note that memory requirements are discussed in Section V-A.4.

What was described in section III-D is not incorporated in Figure 9, in order to keep the algorithm more simple. So, Rule 5 is not included in the figure. In case a node tries to send a packet and the receiver is reported dead, then the sender must remove that node from its table, reset its child pointer and set a flag in the packet, as described in section III-D. Then it looks for a new receiver. In the same way, if a packet is received with the packet's flag set, any previous child and parent pointers should be reset.

IV. Implementation issues

In order to be able to implement the proposed algorithm, we assume that each sensor node has the ability to estimate the Direction of Arrival (DoA) of incident electromagnetic waves. DoA measurements can be implemented in a cost effective way on sensor nodes with the use of switched antenna arrays with an accuracy of 5 degrees [14], [15]. In the experimental analysis at Section V we show how this error in estimation affects the performance of GRAViTy.

Knowing the incident angle of arrival, all nodes are able to execute the proposed conditional propagation algorithm. It is therefore assumed that each node can estimate the DoA of data packets received from nearby nodes, and it can estimate the relative direction of the sink D, provided that the sink sends out beacon messages during an initial phase after deployment and these can be captured by all the nodes in the network. An alternative way for nodes to calculate the DoA of messages coming from their neighbors can be used if nodes know their positions in the network by acquiring it from some location service [16], or by computing it using a hash function in a data-centric storage scheme [17].

Another important characteristic of our protocol is the fact that nodes gain information by overhearing packets broadcasted by their neighbors to other nodes. The transmitter and recipient addresses are included in the packet's MAC header, as well as a sequence control field used to uniquely identify packets. No extra communication overhead is required to gain this information. The only extra energy needed is to keep the nodes, which are inside the transmission range of the node transmitting, awake in order to acquire the MAC header of the packet. However, as we have described, this information reduces substantially the routing path, so overall we gain in terms of energy efficiency.

V. Experimental analysis

All experiments were carried out on connected random unit graphs. For each experiment the network was deployed in an area with dimensions $(500, 500)$. Each of the nodes was placed by choosing its coordinates at random in that interval. We have also assumed a collision-free environment to simplify the simulations and gain a quick insight into several major properties of our algorithm.

Parameters that we consider important in defining a networking context in our experiments are network size (number of nodes) and node density (average number of neighbors for each node). Since our deployment area is the same for each experiment, we achieve different node densities by changing the radius of the nodes. Our experiments were designed to test the protocol in terms of distance traversed by packets.

A. Routing first packet

The first round of experiments is intended to evaluate the performance of routing a single packet in an unexplored random topology. For each experiment the base station was defined to be the upper, rightmost node of the topology

and the source be the lower, leftmost node. In this way we maximize the distance to be traversed by the packet to reach the destination. Since the topologies were random, open and closed voids of different sizes were formed and the packet had to explore and bypass them.

1) Impact of network size: We generated 6000 different random topologies for each network size. For each topology we generated a packet that had to be routed from source to destination and computed for each one the *ratio* of the found path to the shortest path. This is calculated as the sum of hop lengths that the packet traversed over the sum of hop lengths of the corresponding shortest path. Therefore, the ratio shows how much longer the resulted path is compared to the shortest path. Figure 10 shows the mean values of the ratio for different network sizes and a fixed density of 8 neighbors on average. For comparison purposes, the corresponding ratio for GPSR is also shown.

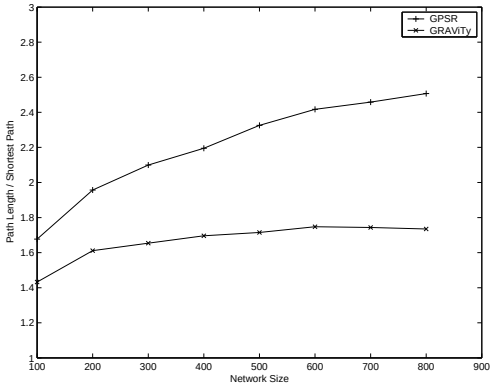

Fig. 10. Ratio of shortest paths found by GPSR and GRAViTy as a function of the network size. The average number of neighbors of each node is 8.

As we see, GRAViTy results in path lengths that are considerably shorter than those of GPSR. As the network size increases, GPSR becomes more inefficient while GRAViTy maintains nearly the same ratio. This is because the impact of dead-ends in the topology can be significant for GPSR as the network size increases. Packets will have to be routed out via longer paths based on the right-hand rule. On the other hand, GRAViTy remains close to the local maximum and explores nearby paths until it finds a way to bypass it.

2) Impact of network density: We next study the effect of different network densities on the average length of routing paths. Our experiments were done on networks of 500 nodes. As the density drops, the sizes of the routing holes increase, the topology becomes more sparse and it is harder for a geographical routing algorithm to find a path to the destination.

Figure 11 shows that in the case of sparse networks, routing paths are considerably longer than shortest paths. As the network becomes more dense, the ratio drops fast and becomes almost 1 for average number of neighbors above 10. In such networks the routing holes are eliminated and both GPSR and GRAViTy manage to find routing paths by just greedy forwarding.

In sparse networks GPSR will have to go into perimeter routing more often which results in longer paths. Likewise,

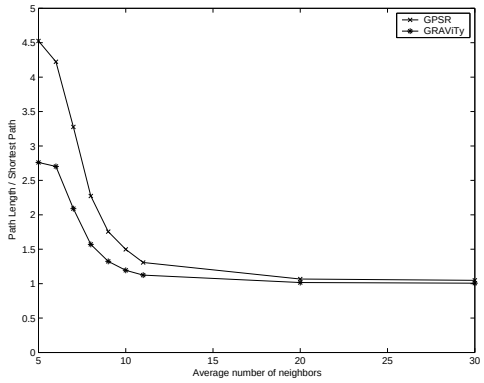

Fig. 11. Ratio of shortest paths found by GPSR and GRAViTy as a function of the network density.

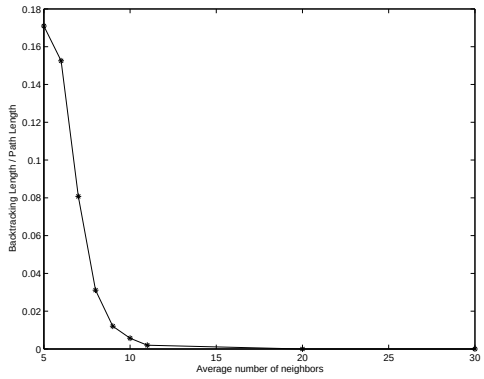

Fig. 12. Average length of backtracking as a function of the network density.

packets routed by GRAViTy will have to traverse larger paths because of backtracking. However, Figure 12 shows that the excess distance traversed because of backtracking is only a small fraction of the overall path. For example, if we assume 8 neighbors on the average for each node, only 3% of the path length is due to backtracking.

3) Impact of direction inaccuracy: So far in our experiments we have assumed ideal antennas. However, as we mentioned in Section IV, realistic switched antenna arrays have an accuracy of ± 5 degrees. So, in Figure 13 we repeated the experiment of the impact of network size on the average length of routing paths, including a statistical error in direction estimation of ± 5 degrees.

This statistical error makes the forwarding procedure probabilistic. The next hop is not always chosen to be the node with the best direction towards the destination. Furthermore, as the network gets larger, there is a higher probability of relative direction inaccuracy. As Figure 13 shows, for small network sizes the error in direction estimation does not affect the performance of the protocol, but for sizes higher then 500 nodes the performance is improved. This perhaps suggests that the use of randomization may further improve the paths found. Recently, other researchers have studied the impact of probabilistic selection of candidate neighbors [18] and have shown that it can also improve the lifetime of the network and decrease the overall end-to-end delay. In future research we indent to study more thoroughly the effect of probabilistic

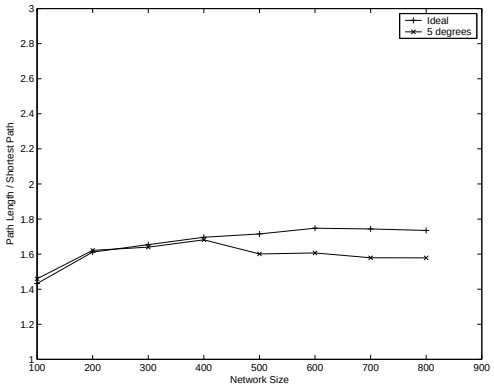

Fig. 13. Direction inaccuracy of directional antennas improves the performance of GRAViTy.

forwarding in GRAViTy.

4) Memory requirements: As we described in section 3, in order to avoid loops, nodes overhear their neighbor's transmissions in order to be aware which of those are part of the routing path. In this way when a node has a packet to forward, it will remove from consideration the neighbors which have forwarded the packet themselves. The required information to be stored is the tuple (packet ID, sender ID), where packet ID is a unique identifier of the packet and sender ID is the ID of the node that sent it.

In order to see how much memory is required by the nodes to store this information, we simulated topologies where several packets were injected in the network at the *same* time. When a packet reached the destination, we deleted from the nodes any entries with the corresponding packet ID and injected in the network a new packet. In this way, there were always a predefined number of packets in the network. Figure 14 shows the results for different network sizes. All of the networks in our experiments had the same density, i.e. the average number of neighbors for each node was 8.

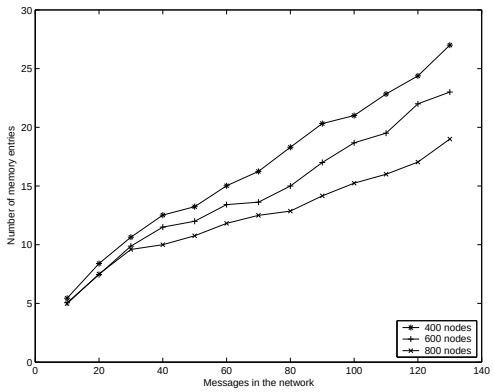

Fig. 14. Average number of entries stored in a node's memory with respect to concurrent packets in the network.

As it can been seen in Figure 14, we reached up to 130 concurrent packets in the network. However, in a realistic sensor network, it is not likely that there will exist so many packets at the same time for two reasons: First, because the nature of the data model in sensor networks is event-based and most of the nodes are asleep except those that connect the source of an event to the destination. Most importantly, however, because data aggregation is applied to intermediate nodes. So, even if multiple events happen at the same time and generate a lot of network traffic, information is aggregated as it reaches the same nodes from different directions.

In our experiments we did not consider any data aggregation. So, in this case, when 10 packets exist in the network, the maximum number of memory entries any node will have to store is 5. When we increase the number of packets by a factor of 10, the number of entries increase by a factor of 3, in the case of network size equal to 800 nodes, and by a factor of 4, in the case of 400 nodes. In any case, we believe that these memory requirements are consistent with the space available in sensor nodes' chips today.

B. Routing subsequent traffic

The routing paths are improved when subsequent packets are routed using child pointers created in the network. Explored portions of the topology can be traversed with no excess hops. In this experimental setting we investigated the performance of GRAViTy on routing 500 packets from random sources in different random topologies of 400 nodes, and show how much the routing path lengths change as more and more packets are injected into the network.

As it is shown in Figure 15, the path lengths rapidly approximate the corresponding shortest path lengths, after a few packets have been injected in the network, and they become only 7% longer than the optimum. In the same figure, the corresponding ratio for GPSR is shown, which does not change as more traffic is routed in the network. GPSR always provides the same paths and does not improve with time.

Since nodes were routing several packets, we added an energy model in order to make the experiment more realistic and simulate the effect of nodes being energy depleted. In this case, GRAViTy employs the mechanism described in Section 6 in order to reset child pointers and adjust to the resulting topology changes.

A free space propagation with data rate set to 2 MB/s is assumed. Packet lengths are 10 Kbit for data packets and 2 Kbit for control packets (RTS/CTS/ACK). Each node has an initial energy of 0.7 Joules. It consumes 660 mW for transmission, 395 mW for reception and 35 mW in the idle state. A node is considered non-functional if its energy level reaches zero.

Figure 16 shows how the ratio of the average path length over the shortest path length is changed, as more packets are routed until the network becomes *disconnected*. In the beginning the ratio drops fast and the performance is substantially improved, as the child pointers are created and packets can follow them to reach the destination. However, as more and more packets are routed, nodes start being energy exhausted and turned off. Then packets need to rediscover the new topological changes and find new routes to the destination. Therefore, the ratio slowly increases as nodes are leaving the network. However, as it can be seen in the figure, the paths remain satisfactory and it is the network that first becomes

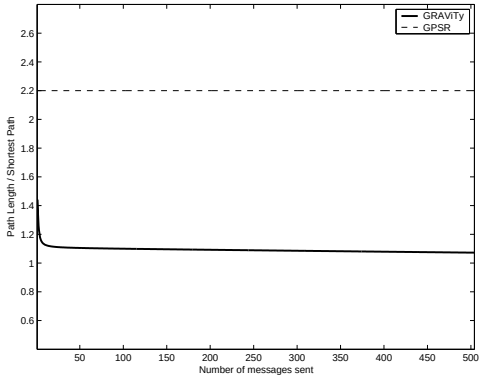

Fig. 15. Average path length as a function of packets routed by the network. The use of past information clearly improves the paths found.

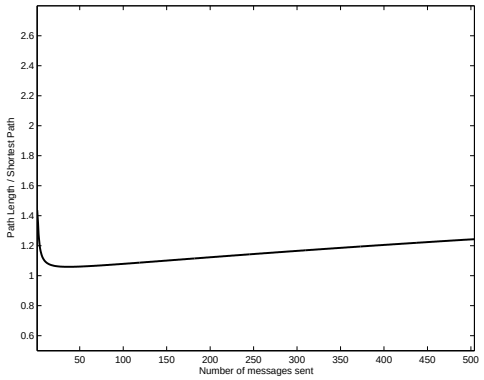

Fig. 16. Impact of node failures to the average length of routing paths.

disconnected before the paths deteriorate a lot. Furthermore, in our examples no aggregation has been used to help keep nodes alive for longer intervals and avoid the frequent path correction induced by the energy depletion of sensors.

VI. Conclusions

In this paper we presented GRAViTy, a protocol for geographic routing with greedy forwarding based on the direction of the node's neighbors and the final destination. The protocol uses only local information, adapts to bad network topologies where voids exist and outperforms GPSR in this respect. To improve path lengths we exploited packet overhearing at the MAC layer that provides useful information to the network layer and help nodes avoid excess transmissions. We believe that cross-layered approaches can help designing efficient protocols for sensor networks and therefore lower layer architectures should provide interfaces that allow some level of customization and transparency.

Our design goal towards a loop-free, single path routing algorithm that guarantees delivery has proved successful. However, the more general question of developing realistic local routing algorithms remains. In our simulations we observed that inserting a statistical error in direction calculations improved the path lengths. In the future we plan to investigate how other details of a real implementations affect our protocol's performance.

References

[1] G. Finn, "Routing and addressing problems in large metropolitan-scale internetworks," Information Sciences Institute, University of Southern California, Tech. Rep. RR-87-180, March 1987.

[2] P. Bose and P. Morin, "Online routing in triangulations," in *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC)*, 1999, pp. 113–122.

[3] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, August 2000, pp. 243–254.

[4] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wireless Networks*, vol. 7, no. 6, pp. 609–616, November 2001.

[5] E. Kranakis, H. Singh, and J. Urrutia, "Compass routing on geometric networks," in *Proceedings of the 11th Canadian Conference on Computational Geometry*, Vancouver, August 1999, pp. 51–54.

[6] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric ad-hoc routing: Of theory and practice," in *22nd ACM Symposium on Principles of Distributed Computing (PODC)*, July 2003, pp. 63–72.

[7] S. Chen, G. Fan, and J. Cui, "Avoid "void" in geographic routing for data aggregation in sensor networks," *International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC), Special Issue on Wireless Sensor Networks*, 2005.

[8] Q. Fang, J. Gao, and L. Guibas, "Locating and bypassing routing holes in sensor networks," in *The 23rd Conference of the IEEE Communications Society (Infocom)*, Hong Kong, China, March 2004.

[9] R. Jain, A. Puri, and R. Sengupta, "Geographical routing using partial information for wireless ad hoc networks," *IEEE Personal Communication*, vol. 8, pp. 48–57, February 2001.

[10] I. Stojmenovic, M. Russell, and B. Vukojevic, "Depth first search and location based localized routing and QoS routing in wireless networks," in *IEEE International Conference on Parallel Processing*, August 2000, pp. 173–180.

[11] T. Dimitriou, I. Krontiris, F. Nikakis, and P. Spirakis, "SPEED: Scalable protocols for efficient event delivery in sensor networks," in *The 3rd IFIP-TC6 Networking Conference (Networking 04)*, May 2004, pp. 1300–1305.

[12] Z. Haas, "Design methodologies for adaptive and multimedia networks," *IEEE Communication Magazine*, vol. 39, no. 11, pp. 106–107, November 2001.

[13] T. Melodia, D. Pompili, and I. Akyildiz, "On the interdependence of distributed topology control and geographical routing in ad hoc and sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 3, pp. 520–532, March 2005.

[14] T. Dimitriou and A. Kalis, "Efficient delivery of information in sensor networks using smart antennas," in *First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors)*, July 2004, pp. 109–122.

[15] S. Preston, D. Thiel, T. Smith, S. O'Keefe, and J. Liu, "Base-station tracking in mobile communications using a switched parasitic antenna array," *IEEE Transactions on Antennas and Propagation*, vol. 46, no. 6, pp. 841– 844, 1998.

[16] J. Li, J. Jannotti, D. Couto, D. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, August 2000, pp. 120–130.

[17] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A geographic hash table for data-centric storage," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, September 2002, pp. 78–87.

[18] T. Roosta, M. Menzo, and S. Sastry, "Probabilistic geographic routing in ad hoc and sensor networks," in *Proceedings of the International Workshop on Wireless Ad-hoc Networks (IWWAN)*, London, UK, May 2005.

Tassos Dimitriou received his BSc degree from the Computer Science and Engineering Department of the University of Patras, Greece in 1991, and his MSc and PhD degrees from the Computer Science Dept. of the University of California, San Diego in 1993 and 1996, respectively. In 1997, he joined the Computer Technology Institute (CTI), where he conducted research on Probabilistic and Approximation Algorithms. In parallel, he was a visiting professor at the Computer Science department of the University of Athens. Since 2001, he is an assistant professor at Athens Information Technology (AIT), leading the Algorithms and Security group where emphasis is given in two distinct areas; the study of secure and energy-efficient protocols for sensor networks and the development of secure applications for networking and electronic commerce. He is a member of IEEE and ACM and a Fulbright fellow.

Ioannis Krontiris received his BSc degree in electronic and computer engineering from the Technical University of Crete, Greece in 2001; and MSc degree in information networking from the Carnegie Mellon University, Pittsburgh, Pennsylvania, in 2004. Since 2005, he has been working with Algorithms and Security group in Athens Information Technology Research Center, Greece, as a research engineer. Since 2006, he has been also pursuing his PhD in sensor networks security. His research interests include wireless security and privacy, routing protocols of sensor networks, embedded systems and distributed computing.