CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Geographical Routing Using Partial Information for Wireless Ad Hoc Networks

Rahul Jain, Anuj Puri, Raja Sengupta
University of California, Berkeley

California PATH Research Report
UCB-ITS-PRR-2001-8

Report for MOU 329

March 2001

ISSN 1055-1425

Geographical Routing Using Partial Information for Wireless Ad Hoc Networks *

Rahul Jain, Anuj Puri and Raja Sengupta
Department of EECS,
University of California, Berkeley
(rjain,anuj,sengupta)@EECS.Berkeley.EDU

May 10,2000

Abstract

In this paper, we present an algorithm for routing in wireless ad hoc networks using information about geographical location of the nodes. We assume each node knows its geographical position and the position of the node to which it wants to send a packet. Initially, the nodes only know their neighbors but over time they discover other nodes in the network. The routing table at a node S is a list $\langle (p_i, S_i) \rangle$ where p_i is a geographical position and S_i is a neighbor of node S. When node S receives a packet for a node D at position $pos(D)$, it finds the p_i in its routing table which is closest to $pos(D)$ and forwards the packet to the neighbor S_i. We prove the correctness of the algorithm and show that our algorithm naturally aggregates the nodes so that the routing table sizes are of size $O(\bar{L}_n \log(n))$, where $\bar{L}_n$ is the mean route discovery path length, and n is the number of nodes. We also present methods for taking positional errors, node failures and mobility into account. We justify the results through simulation.

1 Introduction

A wireless ad hoc network consists of a collection of mobile nodes sharing a wireless channel without any centralized control or established communication backbone. Each node communicates with other nodes within its transmission range. To send a packet to a destination, a node forwards the packet to its neighbor which in turn forwards it to its neighbor and so on, until the packet reaches the destination. The topology of the ad hoc network depends on the location of the mobile nodes and maybe changing with time.

Some of the typical applications of ad hoc networks are in scenarios where setting up a communication infrastructure is difficult (because of mobility) or very expensive (because of terrain). Wireless ad hoc networks can be used in battlefield situations where a communication infrastructure is difficult to build and maintain. Ad-hoc networks are also required for vehicle to vehicle networking in Intelligent Transportation Systems [14, ?]. Other commercial applications include building a wireless access infrastructure such as the one being built by Metricom [?]. Researchers are also exploring the use of ad hoc networks in building networks out of a large number of tiny sensors spread over a geographic area.

In this paper, we will be interested in the routing problem in ad hoc networks. Basic routing algorithms such as link or distance-vector routing require every node to learn about every other node in the network. We refer to this as routing based on *full information.* This is in contrast to routing under *partial information.* In this kind of routing, a node routes based on information about a subset of the nodes in the network. Routing in the Internet provides one example. Routing in the Internet relies on the address hierarchy that mimics the network topology to work correctly. Thus, routing table sizes are manageable and changes in every link of the network do not have to be propagated throughout the network. In flat distance-vector or link-state routing, these savings are not obtained.

Since ad-hoc networks change topology frequently, routing under partial information is of interest for ad-hoc networks. The Zone Routing Protocol [3] is one well known example of an algorithm based on partial information. A node is expected to know the topology in its own zone accurately, and that in other zones only approximately. It is hoped that this will reduce the inter-node communication required to track a changing network topology. Of course, the reduction in the information used for routing may impose other costs. For example, the routes may not be shortest paths.

*This work was supported by the ONR grant N0014-99-1-0695 and by PATH-CALTRANS under MOU 329. It has been published as the technical Memorandum no. UCB/ERL M99/69

This paper presents a new type of distributed, adaptive and asynchronous routing algorithm for ad-hoc networks. It routes based on partial information. It does not rely on any address hierarchy but instead relies on information about node positions, and hence, is called the geographical routing algorithm (GRA). We assume each node knows its own position, and can acquire the position of the packet destination by some means.

Initially, each node only knows about its neighbors. The routing table at a node S is a list $\langle(p_i, s_i)\rangle$ where p_i is a position and s_i is a neighbor of S. When node S receives a packet for destination D, it finds the p_i which is closest to $pos(D)$, the position of D, and forwards the packet to neighbor s_i. The neighbor then repeats the same procedure. In this way, the packet makes its way to destination D. But sometimes when routing a packet, node S may discover that it is closer to the destination than any other position p_i. In this case we say the packet is "stuck" at S. This causes a route discovery protocol to be started. The route discovery protocol finds a path from S to D and updates the routing table of the node k_i on the path by placing the entry $(pos(D), k_{i+1})$ in its routing table where k_{i+1} is the node which follows k_i on the path. In this way new routing entries get added to the routing tables. After the route discovery protocol is completed, the stuck packet can be routed from S to D.

We show that the routing table sizes of our nodes remain fairly small - essentially logarithmic in the number of nodes in the network. Most network routing algorithms do not use position information. However, the results in this paper show that the use of such information in ad-hoc network routing could yield large reductions in routing table size and protocol overhead. We show that the GRA has the same basic properties as most other routing algorithms even though it works with partial information. Given an unknown network, nodes will exchange information and converge to a set of routing tables. We also show that once the routing tables have converged, like other routing algorithms, all routes are acyclic. On the negative side, it should be noted that the GRA does not attempt at shortest path routing. It just uses some acyclic route.

We have confined our discussion to static networks so far. In a dynamic network, links will break and form, nodes will join and leave. The number of protocol packets triggered by each such change should be of the same order of magnitude as characterized in Section 6.4. As we develop mobility models for application environments of interest to us we hope to find out whether these overheads are indeed small enough.

In Section 2, we discuss the GRA in relation to other routing algorithms in the literature. Section **3** presents the system model and problem statement. Section 4 describes the geographical routing algorithm. Sections 5 and 6 discuss issues related to position information inaccuracy and inconsistency, and mobility. Section 7 presents simulation results. Section 8 concludes the paper.

2 Literature Review

In the literature, a number of proposals have been made to solve the problem of routing in wireless ad hoc networks [7, 8, 9, 11, 12, 5, 13]. Most of the approaches are based on the source routing and distance-vector routing approaches. The destination sequenced distance vector (DSDV) [10] routing protocol is based on the classical distance vector algorithm which uses the Distributed Bellman-Ford algorithm (DBF) [6]. The algorithm has modifications to avoid the looping problem present in the basic DBF. Formation of cycles are avoided by tagging each routing table entry with a sequence number. Dynamic source routing (DSR) [4] on the other hand is based on source routing, where the source specifies the complete path to the destination in the packet header and each node forwards the packet to the node specified as the next hop in the packet header. Each source maintains a route cache, where it looks for a path to the destination. If such a path is not found then the source initiates a route discovery protocol to discover the route. Most of the approaches in the literature are variants of the two above approaches with some attempting to combine the best of both. For example, in zone routing protocol (ZRP) [3], each node has a "routing zone" which includes the nodes within some specified distance. Each node knows the topology within its routing zone by using DSDV protocol. For out-of-zone destinations, DSR is used. Other existing proposals are based on finding a backbone for the network (MCDS) [1] or attempt to minimize delay (STARA) [2].

3 System Model and Problem Statement

Suppose there are n nodes in a region that want to communicate with each other. Each node using a wireless link can communicate with only a small subset of the nodes that are its *neighbors*. When a node S wants to transmit a packet to a destination D, it transmits to a neighbor, which in turn transmits to its neighbor, and so on, until the packet reaches destination D.

In a wireless network, each node has a trans-receiver that it uses to communicate. The set of nodes with which a node can directly communicate is not fixed but depends on the power used by its radio transmitter. When the power of the radio transmitter is increased, a node can directly communicate with a larger set of nodes (i.e., it has a larger number of neighbors). In this paper, we will assume that the nodes have fixed the optimal power for their trans-receivers and the neighbors with which a node communicates is hence fixed. Then, we can think of the wireless network as a graph $G = (N, E)$ where the nodes are $N = \{1, \ldots, n\}$ and there is an edge $(i, j) \in E$ if i is a neighbor of j in the wireless network. We assume that i is a neighbor of j if and only if j is a neighbor of i in the wireless network. Then, we can think of the links as being symmetric and the resulting graph as undirected. Furthermore, we assume that the power levels of the trans-receivers are chosen so that the resulting graph is connected. We will also assume that there exists a medium access schedule such that each node can transmit at a certain bit rate without interference.

In this paper, we solve the problem of routing using position information. When a node receives a packet for destination D, it must make a routing decision: which neighbor should the packet be forwarded to? We assume the nodes $\{1, \ldots, n\}$ haves names or IP addresses $\{S_1, \ldots, S_n\}$ and are located at positions $\{pos(S_1), \ldots, pos(S_n)\}$. Each node knows its neighbors and its own position. When a node S_i wants to send a packet to node D, it finds out the position using some position look-up service and addresses the packet to position $pos(D)$. The geographical routing algorithm uses only the position information in making its routing decisions. How the lookup service works, and how a node finds its own position and the position of the destination is not the subject of this paper though we discuss it briefly below.

To decide to which neighbor a packet should be forwarded to, a node consults its routing table. A routing table of node S is a list $\{(p_i, S_i)\}$ (containing in general, fewer entries than the total number of nodes in the network) where p_i is a geographical position and S_i is a neighbor of S. When node S receives a packet for destination position D, it finds the position p_i which is closest to D and forwards the packet to neighbor S_i. We assume initially the nodes only know their neighbors and have no knowledge of the topology. The subject of this paper is how the nodes construct their routing tables in an online manner, why the routing algorithm works correctly and what its performance is.

3.1 Finding position information

We assume that each node can find its own position and the position of the destination node. Although how this is done is not the subject of this paper, we sketch out some technologies which make it feasible. Using the global positioning system (GPS), it is now possible for any node to find its geographical position with a small error. GPS receivers are cheaply available and more precise devices using differential GPS are also available. In applications where the IP address is known but the geographical address is not, a separate translation protocol must be used to find the geographical position from the IP address. This could for example be done using a two way paging network where the IP address is broadcast to all nodes and the node with that specific IP address replies back with its geographical position. Again the details of how this is done are beyond the scope of this paper.

4 The Geographical Routing Algorithm

In this section, we describe the geographical routing algorithm which we refer to as the GRA in short. The basic idea behind the algorithm is to use the geographical position of the destination in making routing decisions. Each node only knows about a small number of nodes in the network. It knows more about nodes that are nearer to it than it does about nodes which are further away. When a node has a packet for a destination, it chooses from the nodes it knows about the one which is closest to the destination, and sends the packet on its way to that node. Along the path, a node may know of an even closer node to the destination. The packet then gets redirected to that node. On its way to that node, it may get redirected again, and so on until it reaches the destination.

For example, suppose a packet is to be sent from from New York city to UC Berkeley, CA. Suppose the New York city node "knows" the route to a node in San Francisco, CA. It then routes the packet according to that node. On the way suppose, there is a node that "knows" a better route to Berkeley, CA. It then routes the packet onto the better route. Now, suppose the packet reaches near Bay Area, and a node "knows" an even better route to UC Berkeley. It, then, routes the packet onto this route, and the packet thus reaches the node in UC Berkeley. Thus, the algorithm has an in-built capability of finding better and better routes to the destination as the packet nears the destination, even though the source node "knows" the network topology around the destination very "coarsely".

We now describe the routing algorithm in detail, and prove its correctness by showing that routing tables are cycle-free and that packets reach their destination. We also quantify the performance of the algorithm in terms of the average routing table lengths.

4.1 The Algorithm

Suppose $G = (N, L)$ is the graph corresponding to our wireless network. The algorithm begins with each node initially knowing only about its neighbors. The routing table at a node S is a list $\{(p_i, S_i)\}$ where p_i is a geographical position of some node and S_i is a neighbor of S. When destination D is closest to position p_i in the routing table, node S forwards the packet to neighbor S_i. Each node thus forwards the packet in the same way till the packet reaches the destination.

But sometimes when routing a packet, node S may discover that it is closer to the destination than any other position p_i. In this case we say the packet is "stuck" at S. This causes the "route discovery protocol" to be started. The route discovery protocol finds a path from S to D (say $Path(S, D) = k_0k_1 \ldots k_l$) and updates the routing table of the node k_i on the path by placing the entry $(pos(D), k_{i+1})$ in its routing table. So now each node on the path knows how to get to D. It is in this way that new routing entries get added to the routing tables. After the route discovery protocol is completed, the stuck packet can be routed from S to D.

We next present our routing algorithm in more detail. We introduce the notion of Voronoi views. This is a geometric way of viewing the routing operation. Each entry (p_i, S_i) in the routing table is associated with a region in $\mathbb{R}^2$ so that if the destination of a packet falls in the region, the packet gets routing according to the entry (p_i, S_i).

4.1.1 Voronoi Cells

Let $C_S^t = \{S_1, S_2, .., S_k\}$ be the set of nodes whose geographic locations are known to node S at time t (we assume $S \in C_S^t$). We refer to these nodes as *centers* at node *S*. We use the positions of the centers to partition $\mathbb{R}^2$ into cells so that all packets for positions which fall within a cell are routed similarly. A cell around the center S_i consists of all points that are closer to S_i then any other S_j. We call this the Voronoi cell with center S_i. We then define the Voronoi view of node S as consisting of the Voronoi cells with centers C_s^t. Formally,

Definition 1 (Voronoi cell) *Let $\{S_1, S_2, .., S_k\}$ be any set of points in $\mathbb{R}^2$. A Voronoi cell with center S_i is dejined as*

$$V_S(S_i) = \{z \in \mathbb{R}^2 : |z - pos(S_i)| =$$

$$\min_{1 \leq j \leq k} |z - pos(S_j)|,\ S_j \in C_S^t\}$$

Definition 2 (Voronoi view) *The Voronoi view at node S at time t is*

$$V_S^t = \{V_S^t(S_i) : S_i \in C_S^t\}$$

Example 1 (Voronoi view) *The example below explains the concept of the Voronoi view. In figure 1, node S has nodes* A, *B*, *C*, *D in its routing table as* centers *but not E. Thus, the Voronoi view of S is the tessellation of the network region based on these nodes. Node E does not affect the Voronoi view of S. But if E is destination for some packet at S, then S forwards the packet to the neighbor node D, which happens to be the closest center to E in S's Voronoi view.*

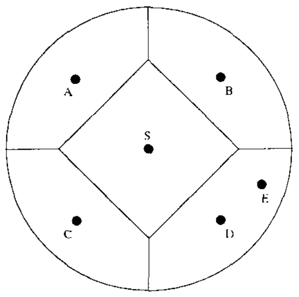


Figure 1: Example of a Voronoi view

Thus, in making a routing decision for a packet going to destination D, node S looks at its routing entries $\{(p_i, S_i)\}$ and finds the position p_j which is closest to D. It then routes the packet to the neighbor of node S forming a Voronoi view based on the centers whose positions are $\{p_1, \ldots, p_k\}$. It then finds the cell in its Voronoi view in which the destination D lies (say p_j), and it then routes the packet to neighbor S_j, as if the packet were meant for the node at position p_j.

4.1.2 Routing Table Structure

The routing table at a node S is structured as shown in figure 2. The first column is the names of nodes that S knows about. We refer to the set of nodes in the first column as the centers at node S. The second column is the positions of the nodes in the first column. We denote this by $pos(S)$. The third column is a column of neighboring node names. Thus if S' is a node in the first column (see 4-th row of figure 2) and N' the node in the neighbor column for S', then packets directed to $V_S(S')$ should be forwarded to N'. Sometimes, we will use the notation $Next_S(S')$ for N', where $Next_S(S')$ is the neighbor of S to which packets for a node in $V_S(S')$ should be forwarded by S. The time-stamp is the time at which the destination node replied to the route discovery message. If the network is mobile, the time-stamp could be used to decide when to obsolete the routing table entry as well.

Some special features of the routing tables are as follows: Since each node is assumed to know its own position, each node has an entry for itself in its own routing table. The first row of figure 2 reflects this. The corresponding neighbor is trivially set to itself. Also, the first column of the routing table should contain all the neighbors of S.

The corresponding entry in the neighbor column would be the neighbor itself.

node	node position	neighbor node	time stamp
S	$pos(S)$	S	T_S
N	$pos(N)$	N	T_N
.	.	.	.
S'	$pos(S')$	N'	$T_{S'}$

Figure 2: Routing table structure

Each routing table entry at S is a 4-tuple $(S_i, pos(S_i), Next_S(S_i), T_{S_i})$. When some of the fields of a routing entry are not of interest, we indicate them with a "-", for example $(-, pos(S_i), Next_S(S_i), -)$. Sometimes, when there is no confusion, we also write this as $(pos(S_i), Next_S(S_i))$.

4.1.3 Packet Format

The packet header has the information shown in figure 3 to aid routing. The source and destination unique names are specified in the packet. The destination position is also specified in the packet. The destination name and position are used for packet forwarding and route discovery. The source time-stamp, source name and source position are included in figure 3 because these may be required in an implementation of the GRA.

destination-name	destination-position
source-time-stamp	source-name
source-Dosition	DATA

Figure 3: Packet format

4.2 Packet Forwarding

Figure 4 describes the packet forwarding algorithm at each node. Suppose a node S receives a packet for destination D. Let C^S denote the set of names of all the nodes that S knows about, i.e., C^S is the set of names in the first column of figure 2. We use $dist(S, D)$ to denote the distance between the nodes S and D, i.e., $dist(S, D) = ||pos(S) - pos(D)||$, and $\preceq_{id}$ to denote the complete order on node names.

The packet forwarding decision is quite simple: At any time, a node knows about only a small subset of the nodes in the network. Initially, this set consists of only the node itself and its immediate neighbors. Later, the nodes that are discovered through the route discovery process are added to its routing table. When a node S receives a packet for destination position D, it finds the entry

```
Node $S$ receives packet for destination $D$ at time $t$:
Let $pos(D) \in V^t_S(S_i)$ for some S, E $C^t_S$

    if (S == $D$)
        // packet reached its destination
    else if ($S_i \neq$ S)
        next-node = $Next_S(S_i)$;
    else
        //packet is stuck
        Initiate route_discovery($S$,$D$);
        next-node = $Next_S(D)$;
```

Figure 4: Packet forwarding algorithm

$(S_i, pos(S_i), Next_S(S_i))$ such that S_i is closer to D then any other S_j. It then routes the packet to $Next_S(S_i)$.

It may turn out that node S is itself closest to D then any other S_j E C^S. In that case, we say that packet is *stuck* and it cannot be forwarded to any of the neighbors according to its current routing table. If the packet is stuck, then node S initiates a route discovery to the destination node D. The route discovery procedure *route_discovery(S,D)* finds an acyclic path $Path(S, D) = \langle k_0, k_1, \ldots, k_l \rangle$ from S to D, and it updates the routing table of node k_i with an entry (D, p_D, k_{i+1}).

It is however possible that a packet destination D is equally close to two nodes S_i and S_j (i.e., $||pos(D) - pos(S_i)|| = ||pos(D) - pos(S_j)||$), and the node lies on the cell boundary. In that case, we assume there is a total order among names, and use that to resolve the tie (i.e., if $S_i \prec_{id} S_j$, the packet is routed to $Next_S(S_i)$, otherwise it is routed to $Next_S(S_j)$).

Example 2 illustrates the GRA routing. Example ?? shows that the use of an order on node-names is important for acyclic routing.

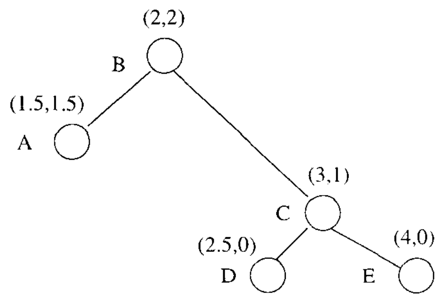


Figure 5: An example network

Example 2 *We illustrate our algorithm on an example network. Consider the network of Figure 5. It con-*

sists of nodes $\{A, B, C, D, E\}$ *which are located at positions* $(1.5, 1.5), (2, 2), (3, 1), (2.5, 0)$ *and* $(4, 0)$ *respectively. The links between the nodes are symmetric and given by* $\{(A, B), (B, C), (C, D), (C, E)\}$.

Initially, each node only "knows" about itself and its neighbors. The initial routing tables at the nodes are shown in Figure 6.

Node	Routing Table
A:	$\{(A, (1.5, 1.5), -), (B, (2, 2), B)\}$
B:	$\{(B, (2, 2), -), (A, (1.5, 1.5), A), (C, (3, 1), C)\}$
C:	$\{(C, (3, 1), -), (B, (2, 2), B), (D, (2.5, 0), D), (E, (4, 0), E)\}$
D:	$\{(D, (2.5, 0), -), (C, (3, 1), C)\}$
E:	$\{(E, (4, 0), -), (C, (3, 1), C)\}$

Figure 6: Initial Routing Tables

Suppose node A gets a packet for destination C *located at* $pos(C) = (3, 1)$. *Node* A *then looks into its routing table and finds that* $pos(B)$ *is closer to* $pos(C)$ *then* $pos(A)$. *So it forwards the packet to node B. Similarly, node B looks at its routing table and finds that* $pos(C)$ *is closer to* $pos(C)$ *than either* $pos(A)$ *or* $pos(B)$. *So it forwards the packet to node* C *which is the destination.*

Next, suppose A *gets a packet for destination D located at* $pos(D) = (2.5, 0)$. *Node A looks into its routing table and finds that* $pos(A)$ *is closer to* $pos(D)$ *then* $pos(B)$. *So the packet becomes* stuck *at node A. This triggers a route discovery The route discovery process finds the path* $\langle A, B, C, D\rangle$ *to the destination D. In the process it also updates the routing tables of nodes* A, *B and C. The new updated routing tables are shown in Figure 7.* A *forwards the packet for D to B which forwards it to C and C forwards it to D.*

Node	Routing Table
A:	$\{(A, (1.5, 1.5), -), (B, (2, 2), B), (D, (2.5, 0), B)\}$
B:	$\{(B, (2, 2), -), (A, (1.5, 1.5), A), (C, (3, 1), C), (D, (2.5, 0), C)\}$
C:	$\{(C, (3, 1), -), (B, (2, 2), B), (D, (2.5, 0), D), (E, (4, 0), E)\}$
D:	$\{(D, (2.5, 0), -), (C, (3, 1), C)\}$
E:	$\{(E, (4, 0), -), (C, (3, 1), C)\}$

Figure 7: Updated Routing Tables

Next suppose A *gets a packet for destination E located at* $pos(E) = (4, 0)$. *A looking into its routing table finds that* $pos(D)$ *is closer to* $pos(E)$ *then either* $pos(A)$ *or* $pos(B)$. *So it forwards the packet to node B based on the entry* $(D, (2.5, 0), B, -)$ *in its routing table. Similarly B finds that* $pos(E)$ *is closer to* $pos(D)$ *then either* $pos(B)$, $pos(A)$ *or* $pos(C)$. *So it forwards the packet to C based on the entry* $(D, (2.5, 0), C, -)$ *in its routing table. Node C finds that* $pos(E)$ *is closer to* $pos(E)$ *than* $pos(D)$ *or* $pos(C)$, *so it forwards the packet to E based on the entry* $(E, (4, 0), E, -)$.

Thus, A was able to route a packet to E even though it did not have E in its routing table. Our simulutions indicate that in large networks this is frequently the case.

4.3 Route Discovery

Suppose node S gets a packet for destination D. The packet gets *stuck* at node S if the destination lies closer to S than any other cell center at S. This triggers the *route discovery* mechanism. which finds an acyclic path $Path(S, D)$ from S to D.

The only requirement for the route discovery mechanism is that it return an acyclic path to the destination, and that it update the routing tables on that path in an appropriate manner. Suppose the acyclic path found is $Path(S, D) = \langle k_0, k_1, \ldots, k_l\rangle$. We then require that an entry $(D, pos(D), k_{i+1})$ be added to the routing table of node k_i. This is the only requirement to ensure the correctness of the routing algorithm. The mechanism by which this path is found has no consequence on the correctness of the routing algorithm. We next state this required property more formally.

Property 1 (Route Discovery Protocol) *If a packet is stuck at node* S, *then* S *starts a route discovery protocol. The route discovery protocol finds an acyclic path* $Path(S, D) = \langle k_0 k_1 \ldots k_l\rangle$ *and adds an entry* $(D, pos(D), k_{i+1})$ *to* $Table(k_i)$ *for* $0 < i < 1$. *We also require that the route discovery protocol update* $Table(k_{i+1})$ *before* $Table(k_i)$.

Several different algorithms can be used to find a path to the destination. Examples of such algorithms are breadth-first search (e.g. flooding) or a depth-first search, the A* algorithm or even the Bellman-Ford algorithm. We next briefly describe the distributed implementation of the breadth-first-search and depth-first-search algorithms that satisfy Property 1.

4.3.1 Path-Finding Phase

We next describe the distributed implementation of the breadth first and depth first algorithms that find an acyclic path to the destination D.

Breadth first search

In the *breadth-first-search* algorithm, node S starts the route discovery protocol broadcasting a route discovery packet (RD packet). Each node that receives the RD packet also broadcasts the packet if it has not forwarded the packet before. This ensures that the paths being found by the route-discovery are cycle-free. Each node that broadcasts the packet, puts its name and address in the packet so that the path being traversed by a route discovery packet is retained. If a packet comes back to a node, it is discarded. Eventually, the route discovery process completes. Each packet that reaches D contains an acyclic path from S to D. Multiple such packets may reach D, and hence, D would know of multiple acyclic paths from S to D.

Depth first search

The *depth-first-search* algorithm on the other hand yields only a single acyclic path from node S to destination node D. Each node puts its name and address on the RD packet. It then forwards it to a neighbor who has not seen it before. The neighbor to which a node forwards the packet is one which minimizes a chosen distance metric. One possible choice for the distance metric is the Euclidean distance (as an estimate of the path length). In that case, node X forwards the packet to neighbor node Y for destination node D if

$$Y = \arg\min_{Y \in N_X} d(X, y) + d(y, D)$$

where N_X is the set of neighbors of node X to which it can forward the packet, and $d(X, Z)$ is the Euclidean distance between node X and node Z.

In case a node has no neighbors left to forward the packet to, it removes its name and address from the packet and returns the packet to the node from which it originally received it. Each node also for some time keeps track of RD packets it has seen before. If a RD packet is forwarded to a node which it has seen before, it refuses it.

source S	destination D
position of D, p_D	visited nodes $v(S, D)$
current path $P(S, D)$	time T_i

Figure 8: Route discovery packet structure

Note that the initial Voronoi view of a node includes the node itself and its neighbors only. It is the route discovery mechanism that puts more cell centers in the routing table and makes the Voronoi view more detailed. With sufficient detail, the route discovery process may not be initiated any more at a node. We call such a Voronoi view, a *complete* Voronoi view.

4.3.2 Updating Routing Tables

When the RD packet reaches destination D, it contains an acyclic path $Path(S, D) = \langle k_0, k_1, \ldots, k_l \rangle$ from S to D. Node D then initiates a route update process by sending an ACK packet back along the path $Path(D, S) = \langle k_l, k_{l-1}, \ldots, k_0 \rangle$. On the way back, an entry $(D, pos(D), k_{i+1})$ is added to $Table(k_i)$. Notice that the routing tables are updated in the order required by Property 1.

4.4 Proof of Correctness

In this section, we will prove the correctness of our algorithm. More specifically, we will show that the routing tables do not contain any cycles (i.e., it is not possible for a packet to get into a loop by following the routing algorithm).

Definition 3 (A cycle in routing tables) *We say the routing tables $\{Table(s_i)\}$ contain a cycle provided there is a destination position D and initial node S_0 such that starting from S_0, the packet follows the path $\langle S_0, S_1, \ldots, S_k \rangle$ without getting stuck and $S_k = S_0$.*

Definition 4 (Centers property) *Suppose for every entry $(S, pos(S), B)$ in $Table(A)$, there is also an entry $(S, pos(S), -)$ in $Table(B)$. We then say that $Table(A)$ satisfies the centers property.*

When the routing tables at all nodes satisfy the centers property, we say the network satisfies the centers property. Intuitively, the centers property is saying that each entry $(S, pos(S), B)$ in $Table(A)$ corresponds to a path. The path goes through nodes $A, B, \ldots$ on its way to node S. We next show that the routing tables in GRA always satisfy the centers property.

Lemma 1 (Centers property) *Consider a wireless network $G = (N, L)$ in which the route discovery process satisfies Property 1. Then the centers property is satisfied by the routing tables.*

Proof Initially each node has itself and its neighbors in its routing table. So for each neighbor n of node A, there is an entry $(n, pos(n), n)$ in $Table(A)$. Because thereisalso an entry $(n, pos(n), -)$ in $Table(n)$, the centers property is satisfied.

Now assume that the centers property holds at time t, and an entry $(D, pos(D), B)$ is added to $Table(A)$. New routing entries can only be added by the route discovery process. So assume that the route discovery was initiated by node S for destination D, and a path $Path(S, D) = \langle p_0, p_1, \ldots, p_k \rangle$ was found where $p_i = A$ and $p_{i+1} = B$. Then because of Property 1, there is an entry $(D, pos(D), -)$ in $Table(B)$. Therefore the centers property is satisfied even after a new entry is added to $Table(A)$. ∎

Theorem 1 (Cycle-free property) *Consider a static wireless network $G = (N, L)$ in which the route discovery process satisfies Property 1. Then there are no cycles in the routing tables.*

Proof Because the route discovery process satisfies Property 1, the centers property holds in the routing tables. Now suppose a packet for node D at position d is placed at node S_0. And suppose the packet follows the path $(S_0 S_1, S_2, \ldots)$ where at node S_i, it is routed according to the entry $(D_i, pos(D_i), S_{i+1})$. From the centers property, $(D_i, -, -)$ is in $Table(S_{i+1})$.

Now either $D_{i+1} = D_i$, or $D_{i+1} \neq D_i$. If $D_{i+1} \neq D_i$, then either $||pos(D_{i+1}) - d|| < ||pos(D_i) - d||$, or $||pos(D_{i+1}) - d|| = ||pos(D_i) - d||$ and $D_{i+1} \prec_{id} D_i$. Now suppose there is a cycle $(S_i S_{i+1}, \ldots S_{i+k})$ where $S_i = S_{i+k}$. It cannot be that $D_i = D_{i+1} = \ldots = D_{i+k}$ because that would imply that the route discovery process found a cyclic path violating Property 1. Therefore, either $||pos(D_{i+k}) - d|| < ||pos(D_i) - d||$, or $||pos(D_{i+k}) - d|| = ||pos(D_i) - d||$ and $D_{i+k} \prec_{id} D_i$. But then $S_{i+k} \neq S_i$, a contradiction. Therefore a packet cannot get into a cycle by following the routing tables. ∎

From the above results it follows that once routing tables have converged packets do not loop. Therefore, either the packet reaches its destination or it gets stuck at a node. If the packet gets stuck, then through the route discovery process, a route is found to the destination, and the packet then gets routed to its destination. Hence, the algorithm ensures that the packet reaches the destination.

4.5 Performance of the Algorithm

4.5.1 Convergence of Routing Tables

One of the advantages of our geographical routing algorithm is that a node does not need to have a routing entry for every other node in the network. In fact, as we will show, after some time, no new route discoveries are initiated, and routing is done with each node having only a small number of entries in its routing table. When the routing tables contain enough detail so that packets can not become stuck, we say that the routing tables have converged or the Voronoi views have become *complete*.

Example 3 *Consider the network of Example 2 The reader should check that the updated routing table in Figure 7 is complete. Note that nodes do not contain routing entries for every other node. For example, node E doesn't know about nodes D,A, or B but can still route packets to them.*

It is best to see this idea geometrically. Corresponding to the routing table at a node is its Voronoi view. Consider the Voronoi view of a node S. Suppose that Voronoi cell $V_S(S)$ contains only node S. Then it is not possible for a packet to get stuck at S because a packet for any other node D falls in a cell other than $V_S(S)$. When this is the case for the Voronoi view at every node, packets can not get stuck in the network.

Definition 5 (Complete Voronoi View) *We say the Voronoi view of node S is complete if $V_S(S)$ contains only node S.*

Now suppose V_S (S) contains a node other than S, say node D. Then when a packet arrives for destination D at node S, it will get stuck. This starts a route discovery and node D is added as a center at node S. The new Voronoi cell with center S is smaller and does not contain D. It is by this process that the Voronoi cell with center S becomes smaller and smaller until it eventually contains only node S. At that point the Voronoi view for node S becomes complete.

Example 4 *Figure 9 (a) shows the Voronoi view at node S with centers $\{S, T\}$, and Figure 9 (b) shows the Voronoi view at S after D is added as a cell center.*

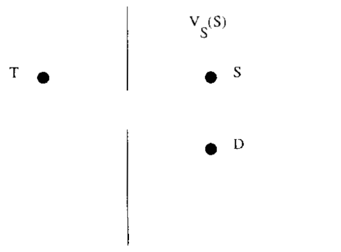
The next lemma states that eventually the Voronoi views at all nodes will become complete.

Lemma 2 (Completion property) *Consider a wireless network $G = (N, L)$ with $V^t = \{V_S^t : S \in \mathcal{N}\}$ being the set of Voronoi views at all nodes of $\mathcal{G}$. Let there be a positive probability of a packet being generated at any source node S for any destination node D in a time interval τ. Then, given any $0 < \epsilon < 1$, there exists a T such that for $\forall t > T$, V_S^t is complete for all $S \in N$ with probability $1 - \epsilon$.*

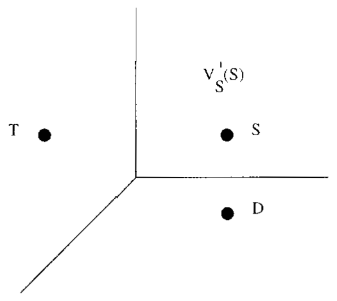
Proof For any $0 < \delta < 1$, there is a T such that node S will generate packets for every other node with probability $1-\delta$ by time T. If a packet for a destination D gets stuck, it is added as a cell center at node S. It follows that by time T, node S will have a complete Voronoi view. Because traffic is generated independently at different nodes, with probability $(1-\delta)^n$, all the Voronoi views at all the nodes will be complete by time T. Now choose δ s.t. $(1-\delta)^n = 1 - \epsilon$. Then for any $0 < \epsilon < 1$, there exists a T such that for $t > T$, V_S^t is complete for all $S \in N$ with probability $1 - \epsilon$. ∎

4.5.2 Size of routing tables of random networks in arbitrarily shaped regions

Claim 1 (Routing table size) *The average routing table size in a n-node network $\mathcal{G}$ when all the nodes have com-*



(a) Voronoi view with centers S and T



(b) Voronoi view with centers S, D and T

Figure 9: Change in Voronoi view on addition of an entry in routing table

plete Voronoi views is $O(\bar{L}\log(n))$ where $\bar{L}$ is the mean route discovery path length.

Let us provide an intuitive justification for this result. Say at node S, the Voronoi cell with center S, contains other nodes, for example, a node D. When a packet arrives for node D, the packet gets stuck, and route discovery process is initiated which causes D to be added as a center at S. This causes the old Voronoi cell $V_S(S)$ to be split (as shown in Figures 9(a) and 9(b)). The new Voronoi cell with center S, $V'_S(S)$, is of smaller size than $V_S(S)$. We are interested in how much smaller is $V'_S(S)$ compared to $V_S(S)$.

Suppose S and D are randomly placed in $V_S(S)$. Then on the average, half of the points in $V_S(S)$ will be closer to S than to D. These points will form $V'_S(S)$. Therefore, on the average $Area(V'_S(S)) \approx \alpha Area(V_S(S))$ where $\alpha = \frac{1}{2}$.

So every time a packet for destination D gets stuck at node S, the node D which was in $V_S(S)$ gets added to S as a cell center, and the area of $V_S(S)$ gets reduced by a factor of a. But this can only be done a certain number of times before S is the only node left in $V_S(S)$, and the Voronoi view of S becomes complete. We are interested in finding the number of times a new cell center can be added at S before the Voronoi view at S becomes complete.

Suppose the nodes are distributed in a region with a unit area. If we form the Voronoi partition based on the nodes in the region, the average area of each cell is $\frac{1}{n}$. So if the number of times $V_S(S)$ gets split is k, then on the average we expect k to satisfy $\alpha^k = \frac{1}{n}$ before $V_S(S)$ contains only node S and the Voronoi view at S becomes complete. This implies that

$$k \approx \frac{logn}{log\frac{1}{\alpha}}.$$

So on the average, packets get stuck $\frac{logn}{log\frac{1}{\alpha}}$ times at a node S before the Voronoi view at S becomes complete.

Now each time a packet for destination D gets stuck at node S, a route discovery process is started. The route discovery returns a path $Path(S,D)$. Let us say the average length of this path is $\bar{L}$ (note that $\bar{L}$ is in fact a function of 71, and hence should be more appropriately written as $\bar{L}_n$). From Property 1, D gets added as a center at every node along the path. So each time a packet gets stuck, $\bar{L}$ new routing entries get added. At each node, packets get stuck $\frac{logn}{log(d,}$ times, and each of these times, I, new routing entries are added to the routing tables. Therefore the average route table size is $O(\bar{L}logn)$.

We have provided an intuitive justification for this result. A more formal argument will be provided in the full paper.

5 Related Issues

5.1 Positional Inaccuracy

Consider a node *i* which thinks it is located at position p_i but which is actually located at p'_i. This could for example happen if node *i* gets its position from GPS and there is an error in the position measurement that it receives from the GPS. Node *i* then advertises its position as p_i and all packets to node *i* are addressed to position p_i even though it is actually located at p'_i. We refer to p_i as the network position of the node since this is what the routing algorithm uses, and to p'_i as the actual position of node *i*. Each packet for node *i* addressed to position p_i either gets to node *i* or gets stuck. If it gets stuck, then route discovery finds a path to node *i*. Although the algorithm works correctly, it can lead to somewhat unmeaningful routing tables as the following example shows.

Example 5 *Consider the network consisting ojnodes A, B, C, D and E. Figure 10 shows their network position, and Figure 11 shows their actual position. The network positions of* A, *B and C match their actual position. But nodes D and E are actually located at positions D' and E'. The links between the nodes are obtained from Figure 11 and are* $\{(E,B),(B,A),(A,C),(C,D)\}$. *Now suppose* **A** *receives a packet for D. So A forwards the*

Figure 10: Network Position

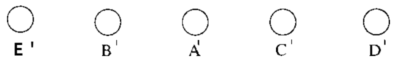


Figure 11: Actual Position

*packet to **B**. But D is actually located at D' and **B** does not have a link to D. So the packet gets stuck at B and a route discovery is initiated. The route discovery finds the path (B,A,C,D) to the actual position D'. A complete routing table for A is $\{(E,B),(B,B),(A,A),(C,C),(D,C)\}$.*

Suppose the error between actual position and network position is 5 (i.e., $||p_i - p_i'|| < 5$). Then if node i is at network position p_i and node j is at network position p_j, then the actual distance between i and j is $||p_i' - p_j'|| < ||p_i - p_j|| + 25$. When a node j receives a packet for position p_i, it can use the bound on $||p_i' - p_j'||$ to decide on its course of action. If the packet gets stuck at j, then j may initiate a route discovery, or it may increase its transmitter power to reach node i.

5.2 Full vs. Partial Route Discovery

When a packet gets stuck at a node X, it initiates a route discovery. Now, the route can be discovered right upto destination node D, or it can be discovered upto a node Y which has node D as a cell center. The first method is called the *full route discovery* and the second method is called the *partial route discovery*. The full route discovery finds a highly reliable and recently updated route to node D. The partial route discovery finds a path to node Y which has D as a cell center. The path from Y to D may have been discovered some time ago and hence may not be as reliable.

5.3 Multiple Route Discoveries

It is possible that at any given time, there are multiple route discoveries going on for the same destination node D, initiated by different nodes. This can result in cycles as the following example shows.

Example 6 *Consider the network of Figure 12. Suppose that a route discovery for destination node D, RD_1, is started by node S_1 at time t_1. Also, suppose that a route discovery for destination node D, RD_2 is started by node S_2 at time t_2. Suppose RD_1 reaches node X_1, which forwards it to node X_2. which then fowards it to node D.*

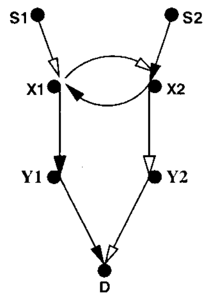


Figure 12: Aynchronous Route Discovery

Similarly, RD_2 reaches node X_2, which directs it to node X_1, which then directs it to node D. Now, suppose that the ACK_1 for RD_1 reaches node X_2, and the routing tables are updated including D as a cell center, and corresponding forwarding neighbor Y_2. Similarly, ACK_2 reaches node X_1, and routing tables are updated at X_1 including D as the cell center; with corresponding neighbor Y_1. Now, suppose that while ACK_1 is traveling from X_2 to X_1, ACK_2 is traveling from X_1 to X_2. The two $ACKs$ then overwrite the entries for D. At node X_1, we then have $N_{X_1}(D) = X_2$, and at node X_2, we have $N_{X_2}(D) = X_1$. Thus, there is a cycle.

This problem can be overcome however, if the destination node time-stamps each route discovery request that it gets. Then, each node that is participating in multiple route discoveries for another node, then updates its routing tables using the RD **ACK** (update) packet with the most recent time-stamp. This does not result in cycles. The proof of this follows exactly the same lines as for Theorem 1.

6 Dynamicity and Mobility in Ad Hoc Networks

In the previous sections, we have assumed that our network is static, and that links and nodes do not fail. We first show with an example that when these assumptions do not hold, the routing tables can become inconsistent and cycles can arise. We then present a simple extension to our algorithm that tries to keep the routing tables consistent in

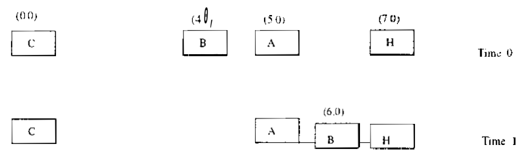


Figure 13: Routing Tables with Cycles in case of inconsistent information

presence of node and link failures.

6.1 Importance of Consistency of Positional Information

So far we have assumed that the nodes in the network do not move. A consequence of this has been that every node that knows about a specific node has the same *consistent* view of it. That is, if node A and B know about node S, then they both believe that S is located at the same network position p_S. As the following example shows, this is an important property.

Example 7 *Consider the example in Figure 13. Nodes A and B are reachable directly from each other. Node C can be reached by A or B, but only via node H. At time 0, B is located at position* $(4, 0)$ *and A's routing table has an entry* $(B, (4, 0))$. *Node B then moves so that at time 1 it is at position* $(6, 0)$. *Node A does not know that B has moved so it still has the old position for B in its routing table.*

Now a packet arrives at node A for node C. Node A forwards this packet to node B because it thinks B is closer to C. B of course is located at position $(6, 0)$ *so it forwards the packet back to A because it thinks A is closer to C. Hence the packet gets into a cycle.*

6.2 Tear Down Protocol

We present a simple extension to our protocol which tries to maintain the centers property and keep the routing tables at nodes consistent. As part of our protocol, nodes need to exchange "hello" messages to discover their neighboring topology. We require that each node also transmit its routing table as part of the "hello" message.

Each node then uses its neighbors' routing tables to check the validity of its own routing table. A node S updates its routing table in one of the following ways:

1. If S receives a "hello" message from node n_i, it puts an entry $(n_i, pos(n_i), n_i)$ in its routing table if it was not already there.

2. If S does not hear from a neighbor n_i for some amount of time, it removes all entries of the form (d_i, p_i, n_i) from its routing table.

3. If $Table(S)$ contains the entry (d_i, p_i, n_i) and S receives $Table(n_i)$ which contains the entry $(d_i, p_j, -)$, then S updates its entry to $(d_i, p_j, n_i, -)$.

4. If $Table(S)$ contains the entry (d_i, p_i, n_i) and S receives $Table(n_i)$ which does not contain an entry $(d_i, -, -)$, then S removes the entry (d_i, p_i, n_i) from its table.

5. After any change to its routing table, S broadcasts the new $Table(S)$.

We refer to the above protocol as the tear down protocol. The reason for this is as follows: suppose there is an entry (s_i, p_i, n_i) in the routing table of S, but node n_i has gone down. Then S deletes the entry $(s_i, p_i, -)$ from its routing table and broadcasts its new routing table to its neighbors. The neighbors in turn do the same. The protocol removes all entries $(s_i, p_i, -)$ in all nodes following which would have taken the packet through the failed node n_i. Alternatively, since the routing entries correspond to paths, all paths which were passing through node n_i get torn down.

6.3 Correctness of the Tear Down Protocol

When nodes or links are going down, it may very well be the case that the "centers" property is violated. Nodes may also have inconsistent views of the network if they are mobile. But once the topology of the network becomes fixed again, the tear down protocol ensures that the "centers" property holds and there are no cycles in the routing tables.

Lemma 3 *Suppose G is a network in which route discoveries are done using full route discovery, and whose topology was changing but has now become fixed. Then after the above protocol runs to completion:*

1. *"Centers" property will hold.*

2. *There will be no inconsistent views in the network.*

3. *There will be no cycles in the routing tables.*

Proof: It can not be the case that there are a sequence of nodes $n_1, \ldots, n_k$ where $n_k = n_1$ and $(s, p, n_{i+1}) \in Table(n_i)$ for $i = 1, \ldots, k-1$ because this would violate Property 1. So when the tear down protocol runs, all entries (s, p, n_i) which do not correspond to a path leading to node s get deleted. Similarly, the correct position of each node gets propagated through the network so that there are no inconsistent views in the network. Because the "centers" property holds after the tear down protocol runs to completion and there are no inconsistent views and no cycles in the routing tables. ■

Hence, tear down protocol tries to maintain the "centers" property and keep the positional information at nodes consistent.

6.4 Overhead due to mobility

In this section, we try to quantify the amount of overhead due *to* mobility. When a node A has a link to node B and node B moves, the link between A and B may be broken. When this happens, the protocol of Section 6.2 communicates this to all nodes which were using this link. This causes all routing entries which were using the link from **A** to B *to* get deleted. Therefore, the amount of overhead is proportional to the number of links that are being broken per unit time. The number of links going down per unit time is directly related to the speed of the nodes. We next try to obtain a formula which quantifies the amount of overhead in terms of the various parameters of the wireless network.

We assume the network has n nodes in a unit area and each node has a transmission radius r.

6.4.1 Overhead from a single link going down

On the average, each node has $n\pi r^2$ neighbors and $cLlog(n)$ entries in its routing table. So on the average $\alpha = \frac{cLlog(n)}{n\pi r^2}$ entries in the routing table of A are using a link from node A *to* a neighbor B. So when the link between **A** and B goes down, α entries in A and cy entries in B become obsolete. This cause $(\frac{2cLlog(n)}{n\pi r^2})\frac{L}{2}$ messages to be broadcast to delete all entries in all nodes which were using the link between A to B.

Since $\frac{2cLlog(n)}{n\pi r^2}$ paths get deleted by each link going down. In steady state, the same number of route discoveries must also be made for each link going down. Each route discovery generates (for example, using breadth first search) n packets. So a total of $\frac{2cLlog(n)}{\pi r^2}$ packets get generated from route discoveries for each link going down.

So each link going down causes

$$\frac{cL^2log(n)}{n\pi r^2} + \frac{2cLlog(n)}{\pi r^2}$$

overhead packets to be generated. That is $O(\frac{Llog(n)}{r^2})$ packets get generated for each link going down.

6.4.2 Number of links going down due to mobility

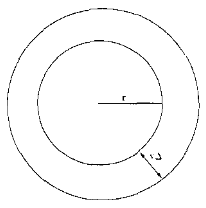


Figure 14: Computing overhead due to mobility

Let us now compute the number of links that go down per unit time. We assume that each node is moving in a random direction at speed v. We will look at a shell of width $v\Delta$ at radius r from a node N. We will be interested in how many of the nodes in the shell move out of node N's range in time A. This is the number of links that will be broken between node N and its neighbor in time Δ.

Figure 14 shows the shell. There are $2\pi rv\Delta n$ nodes in the shell. We are interested in computing the probability that a node in the shell moves out of the circle. This probability is given by

$$p = \frac{1}{v\Delta}\int_0^{v\Delta}\frac{2cos^{-1}(\frac{x}{v\Delta})}{2\pi}dx$$

$$= \frac{1}{\pi}\int_0^1 cos^{-1}(y)dy = \frac{1}{\pi}$$

So for a node N, $2\pi prvn$ links get broken per unit time. Or $O(rvn)$ links get broken per unit time from a single node. Since there are n nodes, a total of $O(rvn^2)$ links get broken per unit time in the network.

6.4.3 Total Overhead

Since $O(\frac{Llog(n)}{r^2}$j packets get generated for each link going down, and $O(rvn^2)$ links get broken per unit time in the network. A total of $O(\frac{Lvn^2log(n)}{r})$ overhead packets get generated in the network per unit time.

7 Simulation Results

In this Section, we describe the simulation framework and results on the performance of the GRA routing algorithm. The performance of a routing algorithm can be measured in terms of the memory requirement at the nodes, and the bandwidth used due *to* the communication overhead. We quantify the performance of the algorithm be simulating the GRA running over random graphs of varying size. In each case, we sample enough random graphs to put our results in a 95% confidence interval.

Our performance measures are the mean routing table size, and the average number of GRA protocol packets generated per node before the routing tables complete. We assume that each protocol packet generated is delivered. Thus the number does not account for retransmissions due to channel variations, medium access control, etc. Note that both measures are independent of underlying link layer or physical layer characteristics. The first measure is related to the memory requirement of the nodes and the second the network bandwidth consumed by the protocol overhead. We have focussed on them to emphasize that the GRA is not tied to a particular link layer protocol or channel type. Its benefits could potentially be realized over many kinds of underlying networks.

7.1 Simulator Description

We generate the random network in two steps. First, the simulator has a graphical user interface that accepts the number of nodes n and the shape of a two dimensional region as input. It then locates n points randomly, with a uniform distribution, in the region. Thus the first step provides a set of node locations. The second step determines the neighbors of each node. We assume that all nodes have the same transmission range and that if the distance between two nodes is less than the transmission range then the two nodes are neighbors, i.e., connected by an edge in the network graph. We find the minimum transmission range such that the nodes form a connected graph. This minimum is found by successive approximation. This process of generating the network graph results in an increase in the average number of neighbors of a node as the node density is increased. This is shown in figure 15.

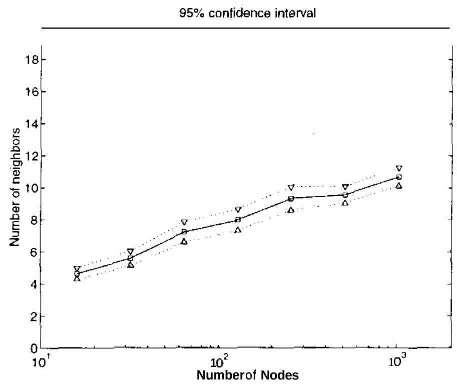


Figure 15: Average number of Neighbors

At each node, there is a routing table to route packets generated or relayed, and a buffer to queue packets. The queue leaks at some constant rate C packets per time unit. The buffer size B at the nodes is large enough so that packets are not dropped.

Packets are generated uniformly randomly U[$\lambda(n)/2, 3\lambda(n)/2$], where

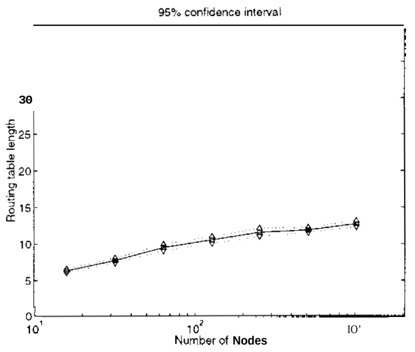
$$\lambda(n) = k\sqrt{n}C$$

is the mean rate at which packets are generated, k is a constant (0.01 in our simulation to prevent buffer overflow). The source-destination pair are chosen randomly. On being generated, a packet gets queued at the node. In each time instant, C (which is 20 in our simulation) packets are forwarded according to the routing table. If a packet is "stuck', it initiates a depth-first-search route discovery, which updates the routing tables upto the destination so that the stuck packet can be routed. The route discovery process is assumed to be instantaneous. We do this to simplify the implementation but nevertheless account for the exact number of path finding and update packets. We assume that all the packets are of same size, and there exists a schedule such that each node can exactly transmit C packets per unit time. Note, however, the performance measures we present are independent of these assumptions, as long as each node is equally likely to originate its next packet for any other node in the network. Nodes may represent agent teams that are located close to each other. For such applications, we think the performance of GRA would be better than under the assumption we make here.

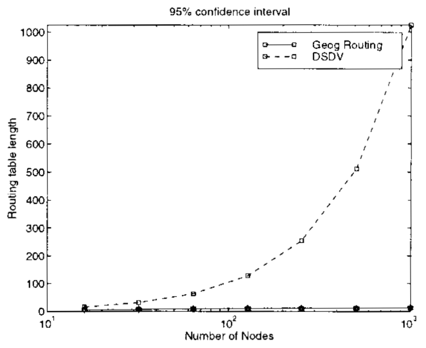
7.2 Results

Figure 16 shows that the mean routing table size is small. In fact, for a 1024 node network, the mean routing table length is only 12.1, The plots show the 95% confidence interval for the mean with 50 simulation experiments. As expected, it grows with the size of the network. Some of this growth is simply the growth in the number of neighbors. Figure 16 plots the two together. We see that most of the growth is accounted for by the increase in neighbors. The increase in the number of non-neighbor remote nodes in the routing table is quite small. This is also as expected because as the number of neighbors of a node increase, it becomes less likely that packets will get stuck at the node. The logarithmic growth in routing table size is in sharp contrast to the linear growth of most ad-hoc network routing algorithms. Figure 16(b) compares the mean routing table length of the GRA routing algorithm with the destination sequenced distance vector (DSDV) routing algorithm. Other algorithms based on distance vector, link-state and source routing also have similar routing table lengths.

Figure 17 (a) shows that the small routing table sizes are, in fact, achieved at very little communication overhead. The overhead in communication is because of the bandwidth used due to the route discovery packets and the updates. However, the update packets are very small in size as compared to the route discovery packets and can also be piggy-backed on other packets, and hence are ignored in our results. We count the number of packets a route discovery transmitted as the communication overhead due to a single route discovery. Figure 17 (a) shows that geographical routing algorithm in a non-mobile network, achieves complete routing tables with communication overhead of less than two route discovery packets per

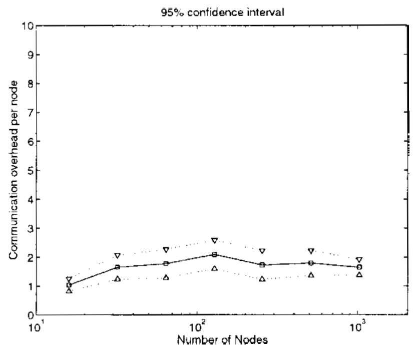


(a) Mean routing table size for GRA

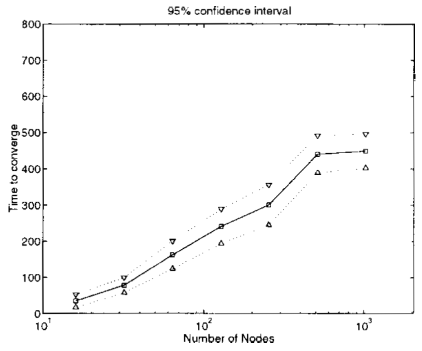


(b) Comparison of GRA and DSDV

Figure 16: Mean routing table size



(a) GRA protocol packets per node



(b) Time to converge (in seconds)

Figure 17: Communication overhead and convergence time are also performance measures

node. The average number of protocol packets per node is approximately constant. Therefore the growth in the number of protocol packets is linear in the size of the network.

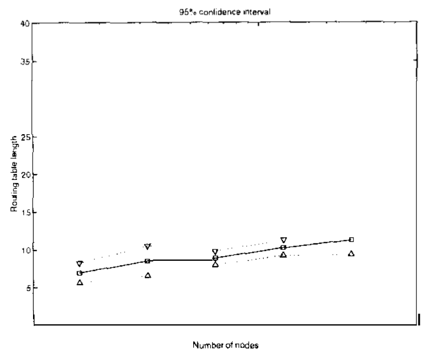
Moreover, as Figure 17(b) shows, with the traffic load as specified above and traffic spread uniformly, the routing tables converge in less than 1000seconds. This means that it takes less than $10C$ packets per node on average for the routing tables of a node to converge. In our simulation C was 20. So, for a 1024 node network, each node generated only 80 packets on average, before it's routing table became complete.

We assumed a random walk model for mobility. The tear down protocol described in Section 6 was implemented in the simulation to take care of mobility. **As** Figures 18(a) and Figure 18(b) show the routing table sizes do not seem to be affected by mobility. In our future work we intend to carry out a comprehensive set of simulations to determine the affect of mobility on communication overhead and throughput.

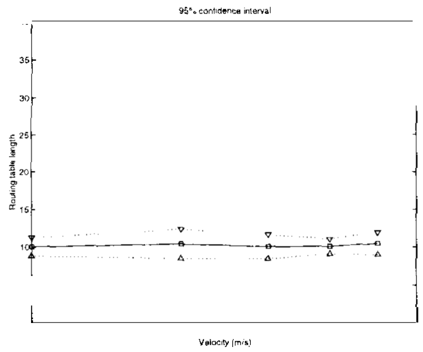
8 Conclusions

In this paper, we have proposed a novel algorithm for routing in wireless ad-hoc networks using geographical information of the nodes. The algorithm is asynchronous, real-time, distributed and scalable. It does not require an architecture or hierarchy to be imposed on the network but provides each node with a distance-dependent aggregated view of the network topology. The basic intuition behind the algorithm is that to route a packet far away from the destination, only a "coarse" knowledge of the network topology is required. As the packet reaches near the destination, nodes in that area are expected to know the topology around the destination in greater detail and will be able to route the packet to the destination.

We showed that if the route discovery process updates routing tables in a particular way, then the routing tables are cycle-free. We also showed that even in mobile networks where the topology changes, the packets may get "stuck" but do not get caught in loops. Further, we quantified the performance of the algorithm in terms of the size of the routing table and communication overhead due to the route discovery process. We presented proposed proto-



(a) Route table size as a function of number of nodes



(b) Route table size as a function of speed

Figure 18: Mobile Networks

cols for handling discovering new nodes, and coping with node failures. These protocols enable the algorithm to handle mobility and dynamicity in network topology.

We showed theoretically and verified through simulation that the algorithm obtains very small routing table sizes and very low communication overhead. Thus, one of the major features of the algorithm is that it is scalable without imposition of any hierarchy (hence ad hoc in true sense). Thus, the algorithm has implications for Internet routing as well. One of the weaknesses of the algorithm is that it assumes an overlaid paging network to provide information about geographical location of the nodes. But with proliferation of GPS receivers, this may not remain an impractical assumption.

We have presented protocols to handle node mobility. Detailed analysis of the algorithm under high mobility and its load balancing properties are subjects of current research. We intend to present those results in future work.

References

[1] B. Das and V. Bharghavan. Routing in ad hoc networks using minimum connected dominating. In *Proc. IEEE International Conference on Communications '97*, 1997.

[2] P. Gupta and P.R. Kumar. A system and traffic dependent adaptive routing algorithm (STARA) for ad hoc networks. In *Proceedings of the CDC*, December 1997.

[3] Z.J. Haas and M.R. Pearlman. The zone routing protocol (ZRP) for ad hoc networks. Internet draft, draft-zone-routing-protocol-00.txt, 1997.

[4] D.B. Johnson and D.A. Maltz. Dynamic source routing (DSR) in ad hoc wireless networks. Internet Draft, 1997.

[5] Y. Ko and N.H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proc. MOBICOM'98*, pages 66–75, August 1998.

[6] M.Steenstrup. *Routing in Communications Networks*. Prentice-Hall Inc., 1995.

[7] J.C. Navas and T.Imelinski. Geocast-geographic addressing and routing. In *Proc. ACM/IEEE MOBICOM'97*, volume 3, pages 66–76, 1997.

[8] V. Park and S. Corson. Temporally-ordered routing algorithm (TORA) version I functional specification. Internet draft, draft-ietf-manet-tora-spec-00.txt.

[9] C. Perkins. Ad hoc on demand distance vector (AODV) routing. Internet draft, draft-ietf-manet-aodv-00.txt, 1997.

[IO] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector (DSDV) routing for mobile computers. In *Proc. SIGCOMM'94*, pages 234–244, August 1994.

[11] V. Rodoplu and T. Meng. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1333–1344, August 1999.

[12] K. Scott and N. Bambos. Routing and channel assignment for low power transmission in PCS. In *Proc. ICUPC, Fifth Intl. Conf. Universal Personal Communications*, volume 2, pages 498–502, October 1996.

[13] S. Singh and M. Woo. Power-aware routing in mobile ad hoc networks. pages 181–190, August 1998.

[14] P. Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.