A Scalable Location Service for Geographic Ad Hoc Routing

by

Jinyang Li

B.S., Computer Science
National University of Singapore, 1998

Submitted
to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2001

Author . . . .
Department of Electrical Engineering and Computer Science
January, 2001

Certified by . . . .
Robert Morris
Assistant Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Scalable Location Service for Geographic

Ad Hoc Routing

by

Jinyang Li

Submitted to the Department of Electrical Engineering and Computer Science
on January, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

GLS is a new distributed location service which tracks mobile node locations. GLS combined with geographic forwarding allows the construction of ad hoc mobile networks that scale to a larger number of nodes than possible with previous work. GLS is decentralized and runs on the mobile nodes themselves, requiring no fixed infrastructure. Each mobile node periodically updates a small set of other nodes (its location servers) with its current location. A node sends its position updates to its location servers without knowing their actual identities, assisted by a predefined ordering of node identifiers and a predefined geographic hierarchy. Queries for a mobile node's location also use the predefined identifier ordering and spatial hierarchy to find a location server for that node.

Experiments using the *ns* simulator for up to 600 mobile nodes show that the storage and bandwidth requirements of GLS grow slowly with the size of the network. Furthermore, GLS tolerates node failures well: each failure has only a limited effect and query performance degrades gracefully as nodes fail and restart. The query performance of GLS is also relatively insensitive to node speeds. Simple geographic forwarding combined with GLS compares favorably with Dynamic Source Routing (DSR): in larger networks (over 200 nodes) our approach delivers more packets, but consumes fewer network resources.

Thesis Supervisor: Robert Morris
Title: Assistant Professor

Acknowledgments

This thesis is the result of joint efforts of many people including Robert Morris, John Jannotti, Douglas De Couto, David Karger and Frans Kaashoek. Robert Morris suggested using geographic forwarding and a distributed location service for ad hoc routing. The original GLS algorithm was proposed by David Karger. Yan Zhang helped me a lot during the initial stage of protocol development and simulations.

I thank my advisor, Robert Morris, for his help throughout the project and for sharing his ideas openly with me. PDOS is a great group to be in. Eddie Kohler and David Mazières are my favorate officemates. Chuck Blake is always there to help.

I also thank Xiaowei Yang for cheering me up at bad times.

for my family

Contents

List of Figures

Chapter 1

Introduction

This thesis considers the problem of routing in large ad hoc networks of mobile hosts. Ad hoc networks are also sometimes referred to as mobile multi-hop wireless networks. Mobile node in the network form their own cooperative routing infrastructure by relaying each other's packets toward their ultimate destinations.

This thesis describes a system, *Grid*, that combines geographic forwarding with a scalable distributed location service to implement routing in a large ad hoc network. We analyze *Grid*'s location service (GLS), show that it is correct and efficient, and present simulation results supporting our analysis.

1.1 Motivation

It is possible to construct large networks of fixed nodes today. Prominent examples include the telephone system and the Internet. The cellular telephone network shows how these wired networks can be extended to include large numbers of mobile nodes. However, these networks require a large up-front investment in fixed infrastructure before they are useful—central offices, trunks, and local loops in the case of the telephone system, radio towers for the cellular network. Furthermore, upgrading these network infrastructures to meet increasing bandwidth demands has proven expensive and slow.

The fact that large fixed communication infrastructures already exist might seem to limit the usefulness of any competing approach. There are, however, a number of situations in which ad hoc networks are desirable. Users may be so sparse or dense that the appropriate level of fixed infrastructure is not an economical investment. Sometimes fixed infrastructure exists but cannot be relied upon, such as during disaster recovery, battlefields. Finally, existing services may not provide adequate service, or may be too expensive. For example, ad hoc networks can be used to extend base station's coverage and overall system capacity. Also, people can setup a large scale cooperative wireless network in their community without relying on any 0 companies to wire them up.

Though ad hoc networks are attractive, they are more difficult to implement than fixed networks. Fixed networks take advantage of their static nature in two ways. First, they 1 distribute network topology information among the nodes, and each node computes routes through that topology using relatively inexpensive algorithms. Since the network topology does not change very frequently due to the static nature of fixed networks, such topology distribution has acceptable protocol message overhead and has shown to be cost effective. Second, fixed networks define route hierarchy and embed routing hints in node addresses because the complete topology of a large network is too unwieldy to process or distribute globally. Neither of these techniques works well for networks with mobile nodes because movement invalidates topology information and permanent node addresses cannot include dynamic location information. However, there is a topological assumption that works well for radio-based ad hoc networks: nodes that are physically close also tend to be close in the network topology; that is, they will likely be connected by a small number of radio hops.

Grid uses geographical forwarding to take advantage of the similarity between physical and network proximity. In order to send packets, the source node first obtains location information (i.e. geographical position) of the intended destination and labels each packet with the geographical position of its destination. An intermediate node only needs to know its own position and the positions of nearby nodes; that is enough information to relay each packet through the neighbor that is geographically closest to the ultimate destination. Although *Grid* forwards packets based purely upon local geographic information, it is highly likely that packets are also approaching their destination as measured by the number of remaining hops to the destination. Because nodes only need local information, regardless of the total network size, geographic forwarding is attractive for large-scale networks.

However, to achieve the full functionality of a traditional routing protocol, a system based on geographic forwarding needs a mechanism for source nodes to learn the current positions of destinations. To preserve scalability, this location service must allow queries and updates to be performed using only a handful of messages. Of course, the location service itself must operate using only geographic forwarding.

1.2 Design Goals

There are many possible designs for a location service. For example, whenever destination node's location information is desired, the source can simply flood the network with location query packets. This simplest design violates our goal of scalability because of its excessive flooding messages. Another possible strategy would be to designate a static central location server whose geographic location is well-known and have all nodes send their location updates and queries to this server. This solution is not scalable either. Not only the central location server becomes a single point of failure, but also there is too much load on it and its neighborhood radio network for processing and forwarding all the location update and query packets. Another disadvantage is that no matter how close the destination node is, the source always has to communicate with the (possibly faraway or unreachable due to network partition) location server before it can send packets to the destination.

We have identified the following characteristics of a desirable distributed location service that serve as our design objectives:

1. **The work of maintaining the location service should be spread evenly over the nodes.**
   No node should be a bottleneck. Relying on any central designated location servers not only increases the load of those servers, making them single points of failure, but also will inevitable congest the radio network around those servers. To spread the load, each node should act as a location server for different sets of nodes.

2. **The failure of a node should not affect the reachability of many other nodes.**
   Node failures in ad hoc networks are much more common than in fixed networks due

to battery problem, less powerful mobile devices etc. Therefore, the location service should be fault tolerant. The failure of any particular small set of nodes should not bring down the entire location 1, rather the performance should degrade gracefully as nodes fail. This effectively excludes solutions that depend on centralized location servers.

3. **Queries for the locations of nearby hosts should be satisfied with correspondingly local communication.**
   When source and destination nodes are nearby each other, the location query should be a relatively inexpensive operation and faster than if the nodes are far away from each other. Specifically, if the query is performed by sending out a location query packet, such packet should not travel very far away. Since we expect more communication to happen between nodes that are close to each other, location queries in general would have much less overhead with this property. In addition, this would also allow operation in the face of network partitions.

4. **The per-node storage and communication cost of the location service should grow as a small function of the total number of nodes.**
   To preserve the scalability of geographic forwarding, the location service itself must be scalable. We evaluate scalability of a distributed location service in terms of the per-node storage and communication cost required for running the protocol.

The *Grid Location Service* (GLS) presented in this thesis satisfies all above requirements.

1.3 Thesis Organization

The second chapter of the thesis describes the geographic forwarding technique used in *Grid*. It evaluates the effects of node density in the network on the performance of greedy geographic forwarding. The third chapter describes the *Grid Location Service*, how it realizes the above mentioned properties and proves its correctness. The next chapter describes how *Grid* puts everything together to route a packet. We also examine several optimizations that make *Grid* work well in a mobile environment. We present our performance evaluation of *Grid* by simulations in Chapter 5. Specially, we answer the questions, "how does *Grid*'s protocol overhead grows as the size of the network grows?", "how well does *Grid* perform with moving nodes?", "Does GLS degrade gracefully with node failures?". We also compare *Grid* with one of the best-known on-demand routing protocol, DSR (Dynamic Source Routing) and show that *Grid*'s performance is superior. Chapter 6 describes the related work in ad hoc routing and Chapter 7 concludes with summaries of the thesis' contributions and future work.

Chapter 2

Geographic Forwarding

Geographic forwarding is a routing technique that bases its forwarding decisions on neighboring nodes' geographic locations. In radio networks, it is very likely that two nodes that are geographically close together will also be close together in terms of the routing path. Geographic forwarding makes use of this characteristics of radio networks.

2.1 Geographic Forwarding Overview

We use a simple scheme for geographic forwarding that is similar to Cartesian routing [8]. Each node determines its own geographic position using a mechanism such as GPS [1]; positions consist of latitude and longitude. A node announces its presence, position, and velocity to its neighbors (other nodes within radio range) by broadcasting periodic HELLO packets. Each node maintains a table of its current neighbors' identities and geographic positions. The header of a packet destined for a particular node contains the destination's identity as well as its geographic position. In order to forward a packet toward location *P*, a node consults its neighbor table and greedily chooses the neighbor closest to *P* as the packet's next hop. The packet stops when it reaches the destination.

In the example shown in figure 2.1, when node A wants to send packets to G, it labels each packet with G's geographic location (i.e. latitude and 0). Each intermediate node chooses among its neighbors the node that is the closest to the destination geographically and forwards the packets to it. For example, node C will pick node E as the next hop for

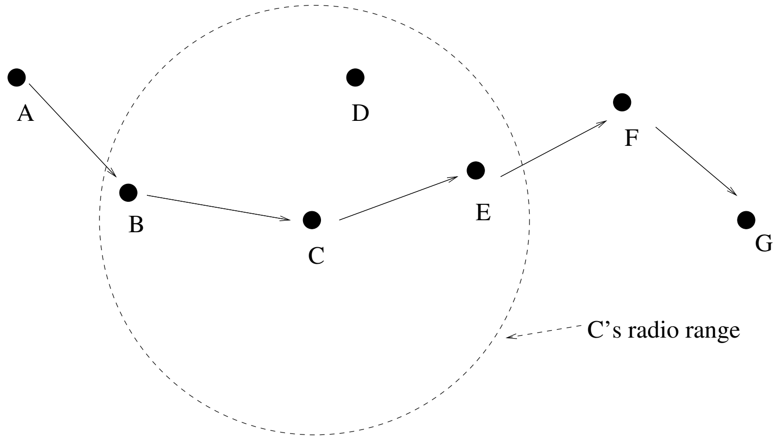

Figure 2-1: An example of a geographic forwarding path. Node A is the source and G the destination.

packets to G.

The greatest advantage of geographic forwarding lies in its simplicity and scalability. No complex routing hierarchy is built yet it is perfectly scalable to big networks, since no matter how big the network becomes, each node only needs to learn about its local neighborhood in order to make forwarding decisions.

Greedy geographic forwarding will fail in the face of adverse topologies. For example, a packet may reach a node that does not know about any nodes closer than itself to the ultimate destination. For example, in figure 2.1, C thinks itself the closest node among its neighbors to G and can not further forward the packets. However, we can see that there is a valid path via $C \to D \to E \to F \to G$. Because D is farther away from G than C, C refuses to forward packets backwards and greedy geographic forwarding will fail. This dead-end indicates that there is a "hole" in the geographic distribution of nodes. In that case, the *Grid* implementation described in this thesis simply gives up and does not attempt to recover from the error. We believe that in truly mobile networks, where topology changes very often, such geographic forwarding failures are temporary and successive retries will succeed after the network "heals" the "hole" itself.

Actively recovering from dead-ends is possible using the same neighbor position information used in geographic forwarding. Karp and Kung propose GPSR [13], a geographic routing system that uses a planar subgraph of the wireless network's graph to route around

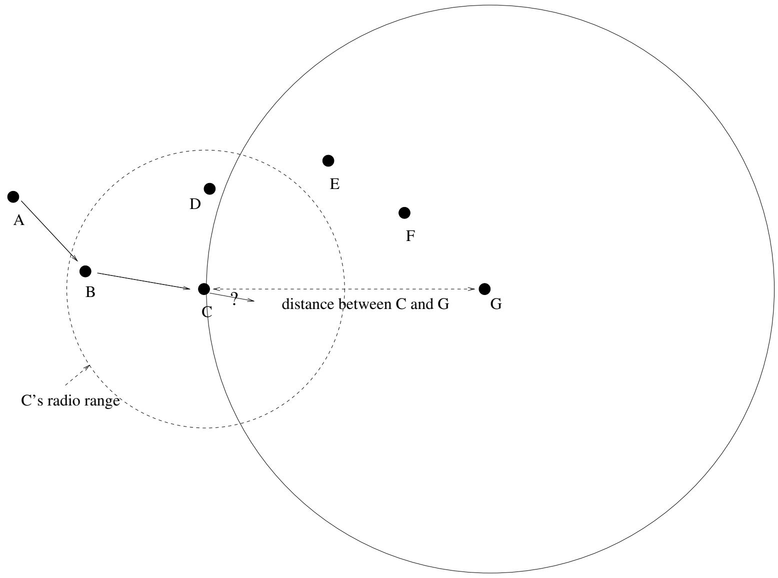

Figure 2-2: Geographic forwarding failure. Node C does not know of any other node that is closer to G than itself, even though there is a valid path ($C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$) to G.

holes. They simulate GPSR on mobile networks with 50–200 nodes, and show that it delivers more packets successfully with lower routing protocol overhead than DSR on networks with more than 50 nodes. Bose et al [3]. independently demonstrate a loop-free method for routing packets around holes using only information local to each node. The method works only for *unit graphs*, in which two nodes can communicate directly in exactly the cases in which they are within some fixed distance of each other.

2.2 Effect of Density

In a random network with uniform nodes distribution, Geographic forwarding works best when nodes are dense enough that dead ends are not common. We present a simple evaluation of the effects of node density using the *ns* [7] network simulator. The simulated nodes have 2 Megabit per second IEEE 802.11 radios [6] with ranges of about 250 meters; each node transmits HELLO messages at 2 second intervals, and routing table entries expire after 4 seconds. Nodes move continuously at $0 \sim 10$m/s; each node moves by selecting a random destination, moving toward it, and selecting a new destination when it reaches the old one. Each node sends packets to three destination nodes selected at ran-

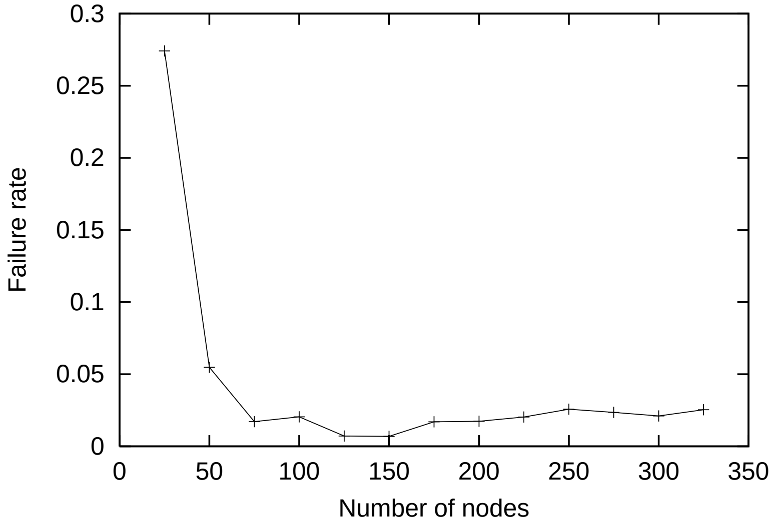

Figure 2-3: Fraction of data packets unable to be delivered using geographic forwarding with a perfect location service, as a function of node density. The simulation area is 1 km^2.

dom; each conversation starts at a time selected randomly over the 300 second life of the simulation. A conversation involves sending 6 packets of 128 bytes each at quarter second intervals. Senders know the correct geographic positions of destinations without the need of a location service in the simulations.

Figure 2-3 is the result of simulations over a range of node densities. In each simulation, the nodes are placed at random in a 1 km^2 square. The graph reports the fraction of packets that were not delivered for each node density. In this scenario, geographic forwarding works well for more than 50 nodes per square kilometer. If 50 nodes are evenly placed in a 1 km^2 square, the inter-node spacing is roughly $141 = 1000/\sqrt{50}$ meters, which is within radio range. More generally, the simulation results agree with a mathematical analysis of random nodes distributed throughout the unit square: one can prove that if the communication radius is r and the number of points exceeds $(6/r^2)\ln(6/r^2)$ per km^2, then dead ends are extremely unlikely to occur.

Chapter 3

The Grid Location Service

Combining geographic forwarding with a mechanism for determining the location of a node implements the traditional network layer: any node can send packets to any other node. In this chapter, we explain the design of a distributed location service, *Grid Location Service*, that makes use of geographic forwarding.

A trivial location service might consist of a statically positioned location server. Nodes would periodically update this server (using geographic forwarding to send update packets to the location server's well-known coordinates) with their current location. For a node **A** to contact node **B**, **A** queries the location server for **B**'s current location before using geographic forwarding to contact **B**.

Using a single location server has a number of problems. The centralized server is a single point of failure; it is unlikely to scale to a large number of mobile nodes; it also causes overload to the centralized server and its neighboring nodes that need to relay location update/query packets to the server; it can not allow multiple network partitions to each function normally in their own partition; and no matter how close nodes are to each other, they must contact a potentially distant location server in order to communicate locally.

The design goal of GLS is to address these problems. GLS is fault-tolerant; there is no dependence on specially designated nodes. GLS scales to large numbers of nodes; our goal is to provide a service that scales to at least the size of a large metropolitan area. Finally, GLS operates effectively even for isolated pockets of nodes. A node should be able to determine the location of any node that it can reach with geographic forwarding. That is, a location lookup should not involve nodes that are too far away from the node performing the lookup and the node being looked up.

3.1 GLS Overview

The main idea of GLS is that every node maintains its current location in a number of *location servers* distributed throughout the network. GLS arranges that different nodes tend to have different sets of location servers. These location servers are not specially designated; each node acts as a location server on behalf of some other nodes. The location servers for a node are relatively dense near the node but sparse farther from node; this ensures that anyone near a destination can use a nearby location server to find the destination, while also limiting the number of location servers for each node. On the other hand long distance queries are not disproportionally penalized: query path lengths stay roughly proportional to data path lengths.

In order for each node to independently select its limited set of location servers, GLS avoids techniques such as leader election or dynamic hierarchy to determine location server responsibility. These schemes place undue stress on the nodes unlucky enough to be elected as a leader or placed at higher levels in the hierarchy. Instead, GLS allows a node **X** to select a set of location servers that, probabilistically, is unlike the set of servers selected by other nodes and does not change drastically as nodes enter or leave the network. Nodes searching for **X** are able to find **X**'s location servers using no prior knowledge beyond node **X**'s ID. This is accomplished by carrying out much the same protocol that **X** used to select its servers in the first place.

Our approach draws its intuition from *Consistent Hashing*, a technique developed to support hierarchical caching of web pages [12]. To avoid making a single node into the bottleneck of the hierarchical cache, that paper used a hash function to build a distinct hierarchy for each page, much as we use a distinct location service hierarchy for each node. Also like our paper, that paper used nested query radii to ensure that queries for a given page did not go to caches much farther away than the page itself.

GLS balances the location server work evenly across all the nodes if nodes themselves and their IDs are uniformly distributed in the network. Such unique, random IDs can be obtained, for example, by using a strong hash function to obtain an ID from a node's unique name. The name could be any uniquely allocated name, such as Internet host names, IP addresses, or MAC addresses. For purposes of discussing the GLS, a node's ID is more interesting than its original name, therefore when we refer to a node **A**, we are referring to the node whose name hashes to **A**.

3.2 Selecting and Querying Location Servers

We present the details of GLS algorithm in this section. The major algorithm of GLS determines how each node selects its set of location servers. With algorithms for location server selection, GLS is able to provide for distributed location lookups by replicating the knowledge of a node's current location at selected nodes. A node **A** hoping to contact node **B** can use the same location selection algorithm to query one of a number of other nodes that know **B**'s location. The fact that the same server selection algorithm is used ensures that **A**'s search for **B**'s location servers and **B**'s original recruitment of location servers ought to lead to the same set of servers.

GLS uses a pre-determined spatial hierarchy to limit the number of location servers for each node and organize the distribution of the servers. At startup, all nodes know the same global partitioning of the world into a hierarchy of grids with squares of increasing size, as shown in Figure 3-1. The smallest square is referred to as an level-0 square. Four level-0 squares make up a level-1 square, and so on. It is important that not every square made up of four level-n squares is also a level-$(n + 1)$ square. Rather, to avoid overlap, a particular level-n square is part of only one level-$(n+1)$ square, not four. This maintains an important invariant: a node is located in exactly one square at each level. This system of increasing square sizes provides a context in which a node selects fewer and fewer location servers at greater distances. Our choice of a grid-based partition is somewhat arbitrary; any other balanced hierarchical partition of the space can be used as well.

Consider how **B** determines which nodes to update with its changing location, using its ID and the predetermined grid hierarchy only. **B** knows that other nodes will want to locate

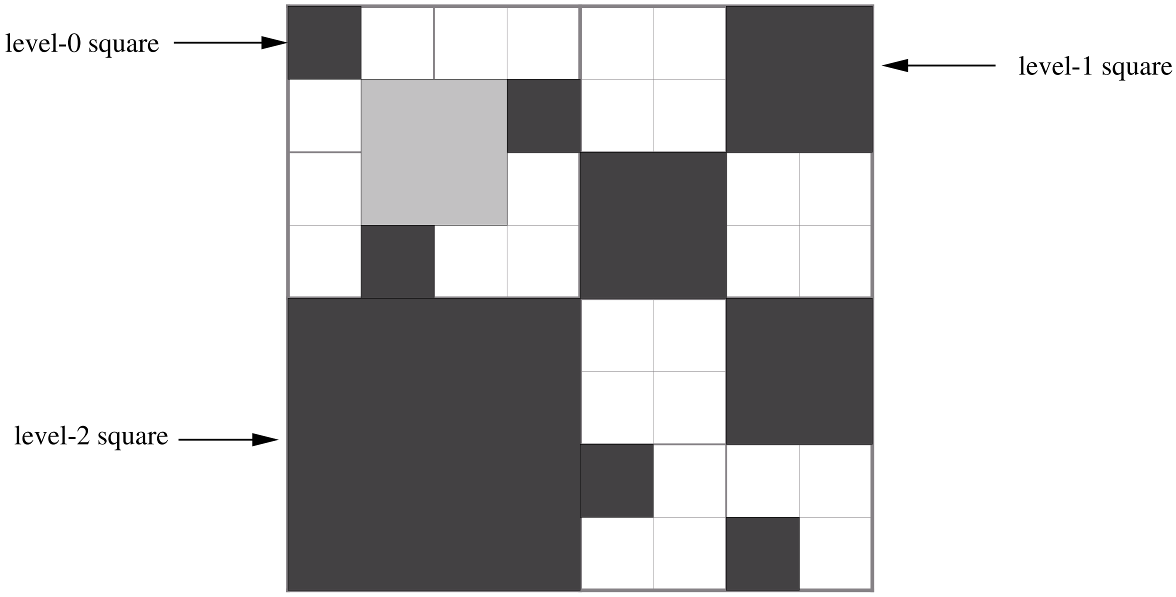

Figure 3-1: GLS's spatial hierarchy. A few example squares at various levels are shown with dark shading. The lightly shaded square is shown as an example of a 2x2 square which is *not* a level-1 square because of its location.

it, but that they will have little knowledge beyond **B**'s ID. **B**'s strategy is to recruit nodes with IDs "close" to its own ID to serve as its location servers. We define the node *closest* to **B** in ID space to be the node with the *least ID greater than* **B**. Such node is also sometimes referred to as **B**'s successor. The ID space is considered to be circular, for example, 2 is closer to 17 than 7 is to 17.

If we consider the tree corresponding to the grid hierarchy, a node selects location servers in each *sibling* of a square that contains the node. The exact details of the selection are best understood with an example (see Figure 3-2). A node chooses a location server in each of the three sibling square at each level of the grid hierarchy. For example, **B** recruits a location server in each of its three sibling level-0 squares, level-1 and level-2 squares. The node chosen to be **B**'s location server is **B**'s successor in that sibling square. In figure 3-2, for example, 23 is **B**(with ID 17)'s location server because it has the closest ID to **B** among all the nodes in that sibling level-0 square (specifically, 23 is considered as "closer" to 17 than 41 in ID space).

Figure 3-3 shows the state of a *Grid* network once all nodes have recruited their location

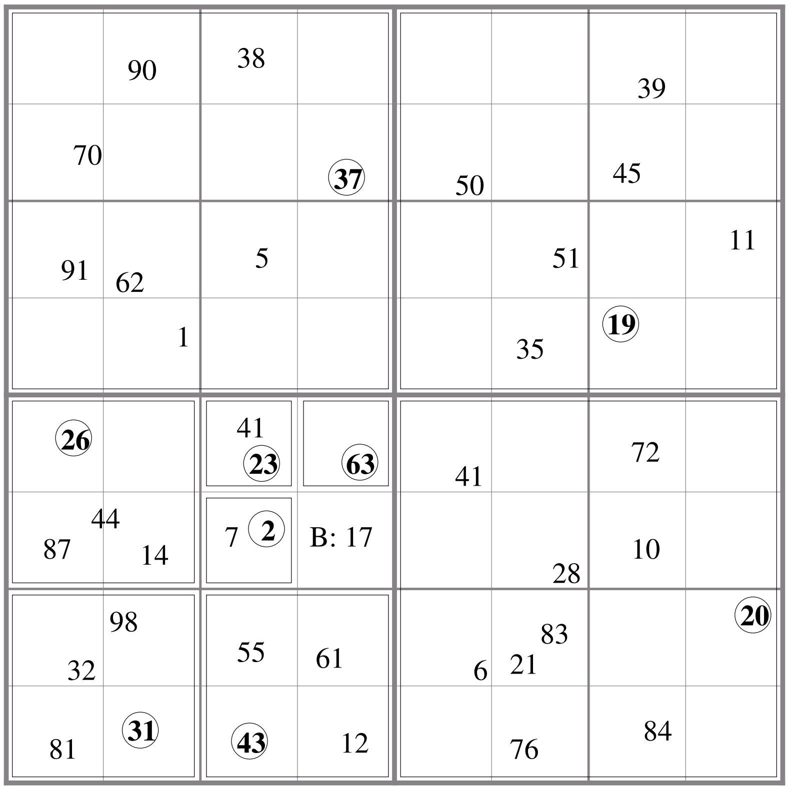

Figure 3-2: The inset squares are regions in which **B** will seek a location server. The nodes that become **B**'s location servers are circled and shown in bold.

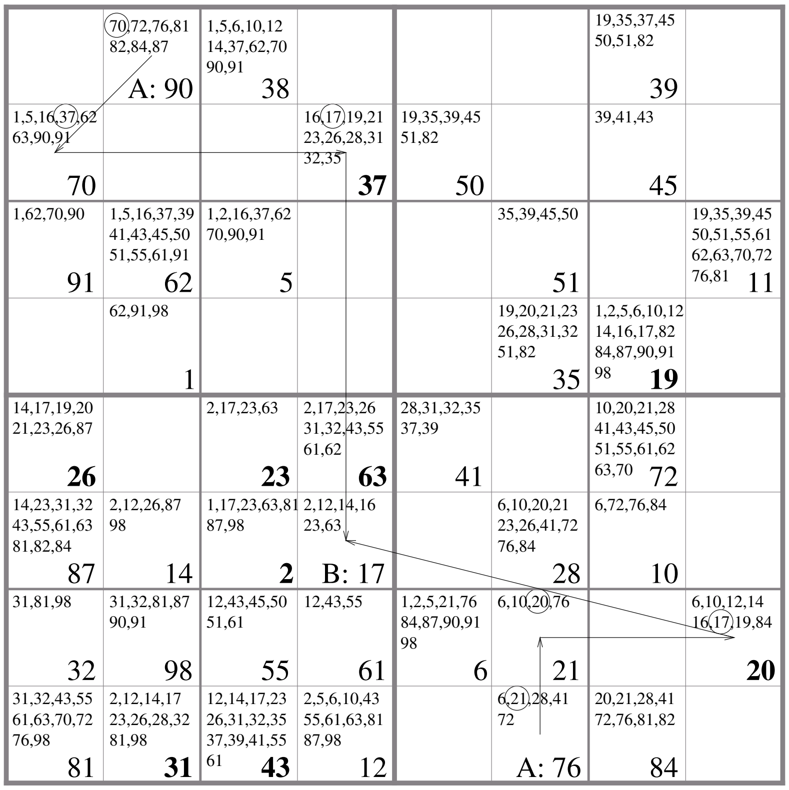

Figure 3-3: An entire network's location server organization. Each node is shown with the list of nodes for which it has up to date location information; **B**'s location servers are shown in bold. Two possible queries by **A** for **B**'s location are shown.

servers using the above explained GLS algorithm. With the complete network state as reference, we can return to the problem of how **A** finds the location of **B**.

To perform a location query, **A** sends a request (using geographic forwarding) to the least node greater than or equal to **B** for which **A** knows (**A** knows about a node if it is the location server for that node). That node relays the query in the same way by forwarding it to the "closest" node it knows and so on. Note that the nodes to whom the location query is forwarded are actually **B**'s successors in increasing levels of squares. Hence eventually, the query will reach a location server of **B** which could 1 the query to **B** itself.

In the above simple illustrations, it appears that we assume that a node can scan an entire square (of arbitrary size) and choose the appropriate nodes (its successors) to act as its location servers. In fact, we allow nodes to route location update packets to their location servers without knowing their identities beforehand. To bootstrap the system, each node should know the identities of every other node in its level-0 square. Assume that a node **B** wishes to recruit a location server in one of its sibling level-n squares, **B** sends a packet, using geographic forwarding, to that square. The first node **L** in the square that receives the packet begins a location update process that closely resembles a query for **B**'s location; but this update will actually carry the current location of **B** along with it. As we will demonstrate in the next section, the update will arrive at **B**'s successor in the level-n square containing **L**. This is exactly the appropriate destination node for the location update to go to; the final destination node simply records **B**'s current location and becomes a location server for **B**.

The only requirement for **B** to distribute its location to the appropriate server in a level-n square is that the nodes contained in the square have already distributed their locations *throughout that square*. If we imagine an entire *Grid* system being turned on at the same time, nodes would need to update their level-1 location servers before they could recruit their level-2 location servers. Once the level-n location servers are operating, there is sufficient routing capability to set up the level-$(n + 1)$ location servers. Although this updating procedure described this way seems to require synchronous operation, synchrony is not strictly necessary. In the actual GLS protocol implementation, we expect to use periodic location updates. We will study GLS performance when nodes join and leave the network frequently in the chapter on performance evaluation. As nodes join and leave the network, updates to location servers are forced to be out of synchrony with each other.

3.3 GLS Efficiency Analysis

We analyze GLS's efficiency in terms of the number of *steps* the location updates and queries take to reach the final location server for a node in a static network. We consider it a single GLS *step* each time a location update/query is forwarded to a different location server. A GLS *step* is distinct from a single *hop* in the geographic forwarding; indeed, each location query step is likely to require several geographic forwarding hops.

In a static network, the number of steps taken by a location query from **A** to **B** is no more than the level of the smallest square in which **A** and **B** are co-located plus 1. In Figure 3-3, the entire diagram is a level-3 square. Therefore all queries can be performed in no more than four location query steps. As we have seen before, location queries use the same location server lookup procedure as location updates, hence the number of steps for a location update to reach its successor in a level-n sibling square is also no more than $n+1$.

At each step, a query makes its way to the destination node's successor (closest in ID space to the destination) node at increasing higher levels of squares in the grid hierarchy. Firstly, the query is forwarded to the best node in the local level-0 square since a node knows all the other nodes in its level-0 square. From this point on, each step moves the query to the successor node in the next larger containing square; when the next larger square contains the destination node, the successor node (closest to the destination ID) must be the destination itself. Thus the query's next step is to the destination. This behavior not only limits the number of steps needed to satisfy a query, it also bounds the geographic region in which the query will propagate. Because the query proceeds into larger and larger squares *that still contain the source*, the query will stay inside the smallest square containing the source and the destination.

To understand why each step brings the query to the destination's successor in a larger square, we will first consider the query from node **A** (76) for the address of **B** (17), shown starting in the lower right of Figure 3-3. Our abbreviated topology has no more than one node per square, so the query trivially begins at the **B**'s successor, itself, at level-0 square. For the next step, the query moves to the **B**'s successor (21) in the level-1 square that contains node **A**, because 76 happens to have the location information for all the nodes in its level-1 square. This is an artifact of our sparse layout. However, we would like to ask, why does 21 know the location of **B**'s 0 in the next higher level square?

Recall that 21 is node **B**'s successor in its level-1 square. This guarantees that no nodes in that square have IDs between 17 and 21. Now, consider a node **X** somewhere in node 21's level-2 square, but not in 21's level-1 square. Recall that **X** also had to choose a location server in node 21's level-1 square (which is actually one of **X**'s sibling square at level-1). If **X**'s ID is between 17 and 21 then **X** *must* have chosen node 21 as a location server since there are no other nodes closer to **X** than node 21 in that level-1 square. Similarly, node 21 knows about *all* nodes in its level-2 square that lie between 17 and itself, including the **B**'s successor. In this case, the **B**'s 0 is 20. At the next step, node 20 must know about all nodes in the level-3 square between 17 and itself. Since nodes 20 and 17 share the same level-3 square (the biggest square in the figure), node 20 knows about node 17, and the query is finished.

The above example demonstrates why node 21 knew node 20's location and was therefore able to move the query from itself which is **B**'s 0 the level-1 square containing **A** to **B**'s 0 in the level-2 square. One may wonder, however, why node 21 does not know about some other node whose ID is between 17 and 20, but which lies at a distant location. This would be undesirable as node 21 would then forward the packet far away simply because, for example, it might know the location of node 19. But this cannot happen because node 20 acts as a shield for node 21 during location server selection. That is, for any nodes closer to **B** outside of the lower right quadrant of figure 3-3, node 21 is guaranteed *not* to be chosen as their location servers; node 20 will always be preferable.

Having built an intuition, we now give an inductive proof that a query needs no more than $n + 1$ location query steps to reach its destination when the source and destination are 1 in a level-n square. Furthermore, the query never leaves the level-n square in which it starts. We assume, without loss of generality, that the destination node's ID is 0. We then proceed inductively to prove the following equivalent claim: in $n + 1$ or fewer location query steps, a query reaches the node with the lowest ID (i.e closest to 0) in the level-n square containing the source. Since the destination is node 0, when the query reaches the level-n square that contains both the source and the destination nodes, it must reach the destination.

Proof:

- *Base case (level-0 square):*

  Suppose the query begins at a node **X**, node **X** may or may not be the node with the lowest ID in its level-0 square. If so, the query trivially reaches the lowest node in the level-0 square after zero location query steps. If **X** is not the node with the lowest ID, then **X** will know the location of the node with the lowest ID in the level-0 square, **Y**, because of our base case assumption that **X** should know every node in its level-0 square. Therefore, the query will be forwarded to the 0's 0 in the level-0 square with at most one step.

- *Inductive step (level-$(n+1)$ square):* If a query is at the node **X** with the lowest ID in its level-n square, then **X** will route the query to the node **Y** with the lowest ID in its level-$(n+1)$ square with one or zero location query steps.

  If **X** has the lowest node ID in the level-$(n+1)$ square, then the claim is trivially true. If not, **X** will know the location of **Y** and *will not* know the coordinates of any node lower than **Y** outside the level-$(n+1)$ square. Node **X** will know the location of **Y** because **Y** will have selected **X** as its location server. Node **Y** must have selected a location server in **X**'s level-n square because **X**'s level-n square is **Y**'s sibling square at level-n. Node **Y** must have selected **X** because **X** is the lowest node in its level-n square and hence is **Y**'s successor that square. Node **X** will not know the location of any node lower than **Y** outside of its level-$(n+1)$ square because when any such node sought a location server in **X**'s level-$(n+1)$ square, Node **Y** was the better choice. Therefore the lowest node that **X** is aware of is **Y** and the query will be forwarded there in one location query step.

□

It is important to note, however, that this proof applies only to a static network. Additional techniques are needed to help *Grid* deal with the problems created by mobility. The next chapter describes *Grid*'s approach to keeping location servers up to date in the face of node motion and *Grid*'s recovery techniques when, despite updates, location information is found to be out of date.

Chapter 4

Grid Protocol Details

This chapter describes the details of the geographic forwarding and GLS protocols that we implemented for simulation purposes. In the previous chapters, most of our algorithm analysis ignored mobility related issues. We will address them more explicitly in the actual protocol implementation here.

4.1 Geographic Forwarding

The geographic forwarding layer uses a two hop distance vector protocol. We prefer this to one hop neighbor table because it gives a node more choices in the next hop selection. This also helps ensure that each node knows the location of all nodes in its level-0 square which is set at about a unit of radio range on a side. Each node maintains a table of immediate neighbors as well as each neighbor's neighbors (2-hop neighbors). Each entry in the table includes the node's ID, location, speed, and a timestamp. Each node periodically broadcasts a list of all neighbors it can reach in one hop, using a HELLO message. When a node receives a neighbor's HELLO message, it updates its local routing table with the HELLO message information. Using this protocol nodes may learn about two hop neighbors—nodes that cannot be reached directly, but can be reached in two hops via the neighbor that sent the HELLO message.

Each entry in the neighbor table expires after a fixed timeout. However, when an entry expires, the node estimates the neighbor's current position using its previously recorded

HELLO
Source ID
Source location
Source speed
Neighbor list: IDs and locations
Forwarding pointers

Figure 4-1: HELLO packet fields.

speed and position. If it would likely still be in range, the entry may still be used for forwarding, but it will not be reported as a neighbor in further HELLO messages. This special treatment is justified by two properties of the 802.11 MAC layer. First, broadcast packets are more likely to be lost in the face of collision than unicast packets. Thus it is not unusual to miss HELLO messages from a node that is still nearby. Second, unicast transmissions are acknowledged. If the neighbor has actually moved away, the transmitting node will be notified when it attempts to forward packets through the missing node. The invalid neighbor entry is then removed immediately and a new forwarding node is chosen.

To select a next hop for geographic forwarding, nodes first choose a set of nodes which are closest to the destination among all nodes in their 2-hop neighbor table. Call this set B. If B contains any single-hop neighbors, remove double-hop neighbors from B. A node, **X**, is then chosen at random from B. If **X** is a single-hop neighbor, the packet is forwarded to **X**, otherwise, since **X** may be reachable from any number of single hop neighbors, the best such neighbor is chosen and the packet is forwarded to that node. If the transmission fails, the chosen node is removed from consideration and the packet is reprocessed, starting with the original B (with **X** removed from its 2-hop neighbor table).

4.2 Updating Location Information

GLS maintains two tables in each node. The location *table* holds the location information for whom the node is acting as a location server; each entry consists of a node ID and that node's geographic location. The location *cache* holds location information that the node learns by looking at update packets it forwards or as a result of a location query. A node only uses the cache when originating data packets. Because each node uses the neighbor table maintained by the geographic forwarding layer to learn about other nodes in its level-0 square, the node does not need to send normal GLS updates within its level-0 square.

The GLS location update is sent periodically in order to cope with node movements. The frequency of periodic updates is determined by the following rules:

- *The faster a node moves, the faster location updates are sent.*

  Intuitively, nodes that move fast change their location information more often. We implemented this rule by associating the location update trigger with the distance a node has 0 since the previous update. For example, if the distance threshold for periodic update is set to be d, a node sends out new update messages to its three sibling level-0 squares every time it has moved d away from where it was when the last updates were sent.

- *Updates are sent less often to distant location servers than nearby location servers.*

  This is necessary to restrict the amount of location update traffic generated in the network. The basic idea behind it is that inaccurate location information about a node far away is potentially more valuable than inaccurate location information about a node that is nearby. In our implementation, each node updates its level-0 squares every time it has moved d unit distance away. In general, a node updates its level-i servers after moving away $2^i d$ unit distance.

Even when nodes are stationary or moving very slowly, they will send out location updates at some maximum interval. This is because even though a node is stationary, its previously updated location servers could move away or fail and the node should send out updates to recruit new servers.

Location update packets (see Figure 4-2) include a timeout value that corresponds to the periodic update interval, allowing the servers to invalidate entries shortly after a new entry is expected. The time at which the location update packet is generated is also included in the update packet so that the freshness of location information for a given node obtained from different servers can be compared.

LOCATION UPDATE
Source ID
Source location
Source timestamp
Update destination square
Update timeout
Next location server's ID
Next location server's location

Figure 4-2: GLS update packet fields.

When forwarding an update, a node adds the update's contents to its location cache. The node associates a relatively short timeout value with the cached entries regardless of the recommended timeout value carried in the update packet.

A location query is only initiated when the sender first wants to communicate with a destination node and has no idea of its whereabout. After the destination node is successfully found using location query and data communication begins, both source and destination nodes keep each other's location information up-to-date by piggybacking their current location on data packets. In the case of one-way communication, the destination node would explicitly send periodic new location information back to the sender.

4.3 Performing Queries

When a node **S** originates a data packet for destination **D**, it first checks its location cache and location table to find **D**'s location. If it finds an entry for **D**, it sends the packets to **D**'s recorded location using geographic forwarding. Otherwise, **S** initiates a location query for **D** using the GLS. GLS tries to deliver the query packet (Figure 4-3) to **D**, which will geographically route a response to **S** that includes **D**'s current location.

If **S** has to initiate a GLS query, it stores the data packet in a send buffer while it waits for the response from **D**. Node **S** 1 the query periodically if it gets no reply, using binary exponential backoff to increase the timeout intervals.

LOCATION QUERY
Source ID
Source location
Ultimate target ID
Next location server's ID
Next location server's location
Timestamp from previous server's database

Figure 4-3: GLS query packet fields.

4.4 Location Query Failures

A location query may fail for two reasons. First, a node may receive a query packet for **D**, and not know the location of any node with an ID closer to **D** than itself. This type of failure is relatively uncommon. It can occur when a location server has not recently received a location update for a node it should know about. Because the server has timed out the node's previous update, it has no way to forward the query packet. There are ways to alleviate these failures, such as using stale location data in a last ditch effort to forward a query packet if the query would otherwise fail. The second type of query failure occurs when a location server forwards a packet to the next closest node's square, but the node is no longer in that square (that is, the location information at the previous location server is out of date). Because this failure mode is more common, *Grid* contains a specialized mechanism to alleviate the problem.

Consider a node **D** that has recently moved from the level-0 square s_1 to the level-0 square s_2. Node **D**'s location servers, particularly those that are far away, will think that **D** is in s_1 until **D**'s next updates reach them. To cope with this, **D** leaves a "forwarding pointer" in s_1 indicating that it has moved to s_2. When a packet arrives in s_1 for **D**, it can be correctly sent on by following the forwarding pointer. **D** broadcasts its forwarding pointer to all nodes in s_1 when leaving. Conceptually, we can think of the forwarding pointers as being located in the *square* s_1 rather than at any particular node. Therefore, all nodes that move into s_1 should pick up the forwarding pointers associated with s_1, and when nodes leave s_1, they should forget the corresponding forwarding pointers. To propagate forwarding pointers to all nodes in the level-0 square and keep all newcomers to the square updated, a randomly chosen subset of the forwarding pointers stored at a node (up to five in our simulation implementation) is piggybacked on the node's periodic HELLO messages. Upon hearing a HELLO message, a node adds each forwarding pointer in that message to its own collection of forwarding pointers, but only if the pointer's original broadcaster was in the same square as the node. In this way, forwarding pointer information is effectively and efficiently spread to every node in the square. With this propagation mechanism, even if all the nodes that originally received **D**'s forwarding pointer were to leave the square themselves, the information would still be available in the square.

Chapter 5

Performance Analysis

This chapter presents simulation results for *Grid* that show how well it scales. Good scaling means that the amount of work each node performs does not rise quickly as a function of the total number of nodes. We use two metrics for evaluating the scalability of GLS: the number of location database entries each node must store, and the number of protocol packets each node must originate or forward. We also compare the performance of *Grid* as a complete ad hoc routing protocol with another popular on-demand routing protocol, DSR [11]. The results show that *Grid* scale well with the number of nodes.

Mobility increases the work required in two ways. First, a node that moves must update its location servers. Second, if a node has moved recently, some nodes may retain out-of-date location information for it; this will cause queries for the moved node to travel farther than necessary, or to fail and need to be re-sent. Handling mobility requires a tradeoff between the bandwidth used by location updates and the bandwidth available for data. If a moving node sends updates aggressively, other nodes are more likely to be able to find it. However, the updates consume bandwidth in competition with data. Worse, a very aggressive update policy may cause enough congestion that updates themselves are dropped. At the other extreme, a node could send updates infrequently even when moving quickly, increasing the amount of bandwidth available to data. However, that bandwidth is not useful if the success rate of location query becomes low because of inaccurate location information. The simulations show that *Grid* can achieve a reasonable tradeoff for the choice of update rate.

5.1 Simulation Scenario

The simulations use CMU's wireless extensions [9] for the *ns* [7] simulator. The nodes use the IEEE 802.11 radio and MAC model provided by the CMU extensions; each radio's range is approximately a disc with a 250 meter radius. The simulations without data traffic use 1 Megabit per second radios; the simulations with data traffic use 2 Megabits per second radios. Each simulation runs for 300 simulated seconds. Each data point presented is an average of five simulation runs.

The nodes are placed at uniformly random locations in a square universe. The size of each simulation's universe is chosen to maintain an average node density of around 100 nodes per square kilometer. One reason for this choice is that we intend the system to be used over relatively large areas such as a campus or city, rather than in concentrated locations such as a conference hall. Another reason is that we expect any deployed system to use radios that allow the power level to be decreased in areas with high node density. The GLS level-0 square is 250 meters on a side. For a network of 600 nodes, which is the biggest simulation we have done, the grid hierarchy goes up to level-4 in a square universe 2900m on a side.

Each node moves using a "random waypoint" model [5]. The node chooses a random destination and moves toward it with a constant speed chosen uniformly between zero and a maximum speed (10 m/s unless noted otherwise). When the node reaches the destination, it chooses a new destination and begins moving toward it immediately. These simulations do not involve a pause time.

5.2 GLS Results

The results in this section involve only GLS (and geographic forwarding), without any data traffic. The default simulation parameters for this section are an 802.11 radio bandwidth of 1 Megabit per second, and a communication model in which each node initiates an average of 15 location queries to random destinations over the course of the 300 second simulation, starting at 30 seconds. The location update threshold distance is an important

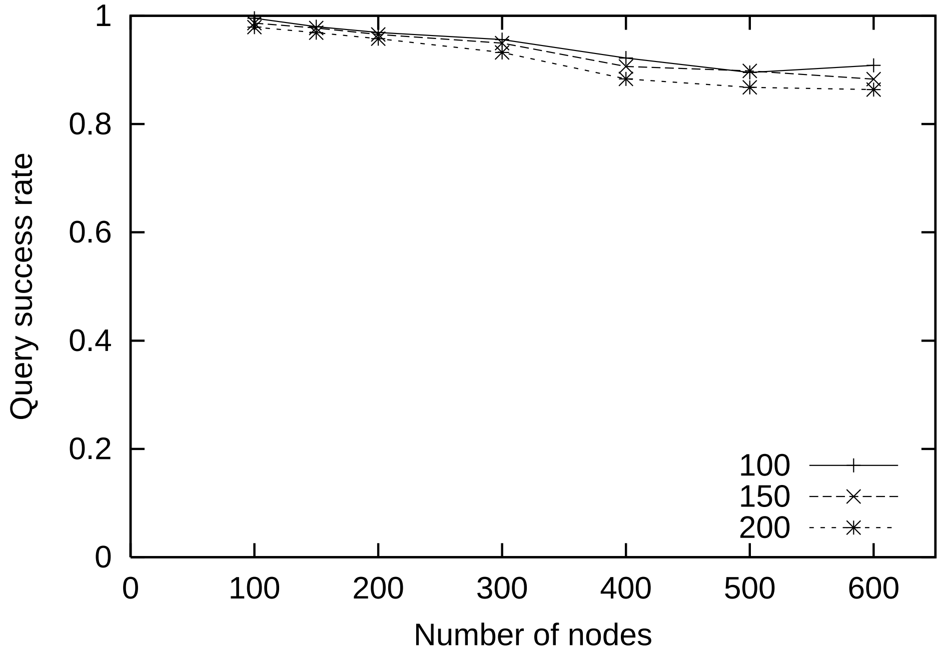

Figure 5-1: GLS query success rate as a function of the total number of nodes. The nodes move at speeds up to 10 m/s (about 22 miles per hour). Each line corresponds to a different movement update threshold.

parameter that may need to be tuned. For this reason we present results for three values of the threshold: 100, 150, and 200 meters.

Figure 5-1 shows the success rate for GLS location queries, as a function of the total number of nodes. Queries are not retransmitted, so a success means a success on the first try. As mentioned earlier, most failures are due to either location information invalidated by node motion or nodes not being correctly updated because of delayed or lost location updates. The success rate for data sent after a successful query would be much higher than indicated here, because the endpoints of a connection directly inform each other of their movements.

Figure 5-2 shows the average number of *Grid* protocol packets forwarded and originated per second per node as a function of the total number of nodes. *Grid* generates three types of protocol packets: HELLO packets that are generated every two seconds but not forwarded, location update packets that are also periodic but require forwarding, and location query and reply packets that also require forwarding. As location updates are generated

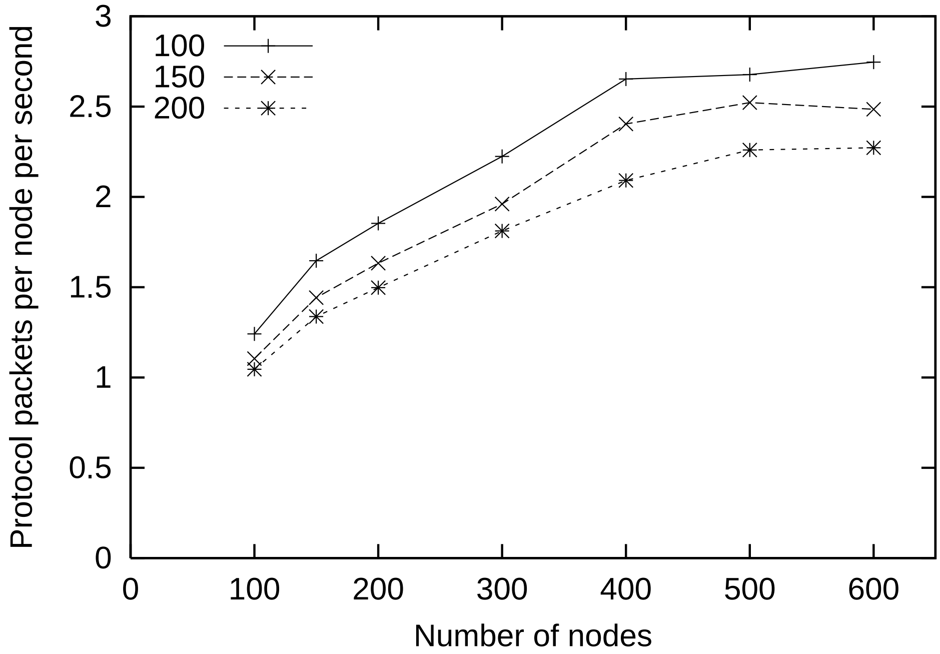

Figure 5-2: Average number of *Grid* protocol packets forwarded and originated per second by each node as a function of the total number of nodes. Nodes move at speeds up to 10 m/s.

by nodes as they move, the results depend on node speeds; the simulated nodes move at speeds uniformly distributed between 0 and 10 m/s. Figure 5-2 is generated from the same simulations that produced Figure 5-1. The graph shows that the *Grid* imposes a modest protocol traffic load as the network size grows.

Figure 5-3 shows how the distance that query packets travel compares with the actual distance in hops between the source and the destination. We record the total number of geographical forwarding hops (for all query steps) that each query takes, as well as how many hops the reply takes. Since query replies are sent directly to the query source using geographic forwarding, the reply return path indicates the geographical forwarding hop distance between the source and destination. We averaged the query hop lengths for all queries with a given response hop length. The graph shows that on average, query packets only travel about 6 hops more than the geographical forwarding route between nodes. Also, the distance traveled by a query between two nodes is proportional to the actual distance between those nodes. Our simulation agrees with a theoretical analysis that proves that

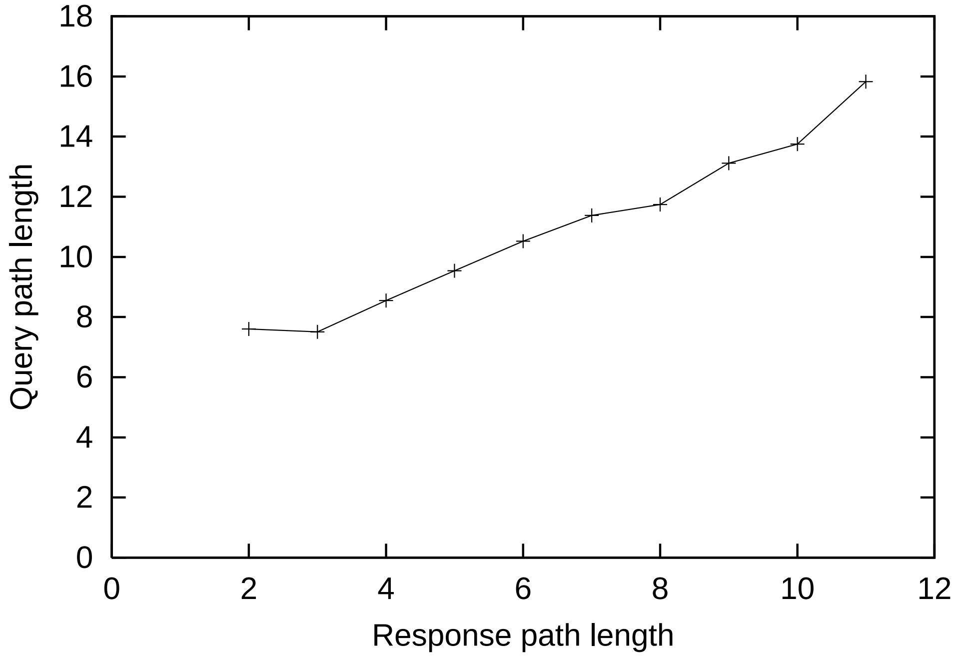

Figure 5-3: Average query path length (in hops) as a function of the query reply path length, for 300 nodes moving up to 10 m/s.

in a sufficiently dense uniform distribution, the number of hops traveled by the query is proportional to the distance to the destination. The simulation involves 300 nodes moving at speeds up to 10 m/s, with a location update threshold of 200 meters.

Figure 5-4 shows the effect of the total number of nodes on the size of each node's GLS location table. The plots include both the average and maximum location table size over all nodes. The spikes at 150 and 400 nodes occur because the simulated area does not exactly fill a hierarchy, causing the database load to be distributed unevenly. At these points, the maximum database size is larger because the squares that extend across the edge of the simulated area contain relatively few nodes; these nodes must store more than their fair share of location database entries. On the other hand, the average table size grows very slowly with the network size.

This highlights a problem that may arise in practice when nodes are not uniformly distributed. A small number of nodes in a high-level square may end up responsible for tracking the locations of a large number of nodes in sibling squares. This would require large amounts of space in these few nodes.

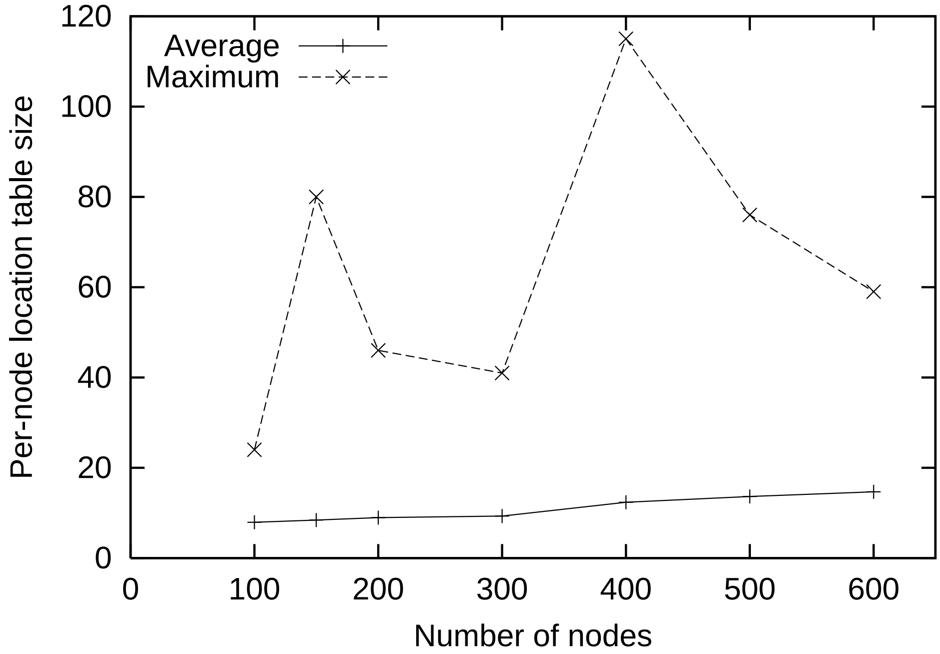

Figure 5-4: Average and maximum per-node location database size (number of entries) as a function of the total number of nodes. The nodes move at speeds up to 10 m/s.

Figure 5-5 shows the effect of node movement speed on the GLS query success rate, for 100 nodes. As nodes move faster, their location servers are more likely to be out of date. On the other hand, the nodes also generate updates faster. The net effect is that the query success rate is relatively insensitive to node speed, however, the update traffic grows as nodes move faster.

Figure 5-6 shows the effect of nodes turning on and off. Some nodes are always on, while the rest alternate being on and off for intervals uniformly distributed from 0 to 120 and 0 to 60 seconds, respectively. As we are simulating node crashes, nodes do not do anything special before turning off; they simply lose all their location table data. In practice, if a node is just turned off normally, GLS should allow it to redistribute its location table to get better performance. Each point in the graph represents a simulation in which a different fraction of nodes are always on. The simulations involve 100 nodes, each moving with a maximum speed of 10 m/s. The statistics are limited to queries addressed to nodes that are turned on; no queries are generated to nodes that are off as these queries will always fail. When a node turns off, a part of the distributed location database is lost; when a node turns

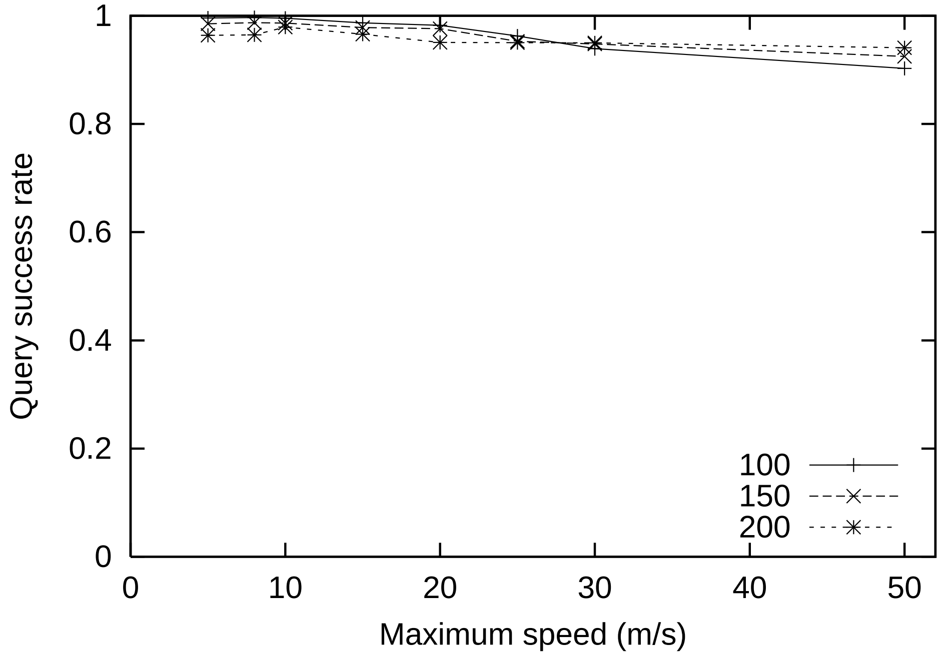

Figure 5-5: GLS query success rate as a function of maximum node speed in a network of 100 nodes. 50 m/s is about 110 mph.

on, it will not be able to participate correctly in the update and query protocol for a while. The graph shows that even a great deal of instability does not have a disastrous effect, and that the query success rate degrades gracefully as nodes turn on and off.

5.3 Data Traffic

The simulations in this section measure *Grid*'s behavior when forwarding data traffic. The 802.11 radio bandwidth is 2 Megabits per second, and the location update threshold distance is 200 meters. The data traffic is generated by a number of constant bit rate connections equal to half the number of nodes. No node is a source in more than one connection and no node is a destination in more than three connections. For each connection four 128-byte data packets are sent per second for 20 seconds. Connections are initiated at random times between 30 and 280 seconds into the simulation. For purposes of comparison we include results for the DSR [11] protocol. This may not be a fair comparison since DSR is optimized for relatively small networks [4].

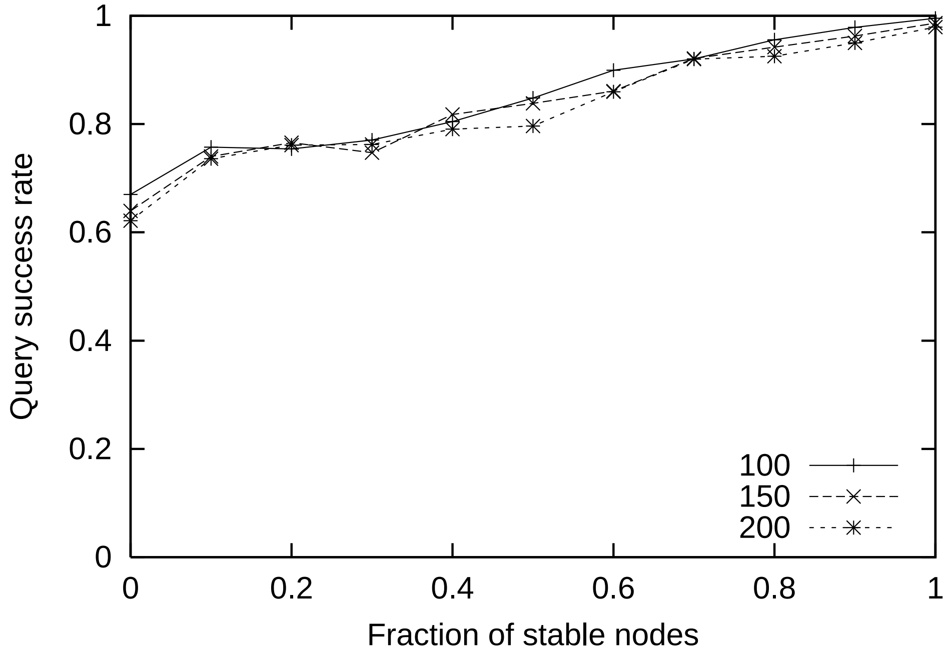

Figure 5-6: The effect of unstable nodes on the query success rate. The X axis indicates the fraction of nodes that are always on; the remaining nodes cycle on and off for random periods up to 120 and 60 seconds, respectively. The network consists of 100 nodes moving at speeds up to 10 m/s.

Figure 5-7 shows the fraction of data packets successfully delivered. Most of the data packets that *Grid* fails to deliver are due to GLS query failures; these packets never leave the source. Once *Grid* finds the location of a destination, data losses are unlikely, since geographic forwarding adapts well to the motion of intermediate nodes. Below 400 nodes, most of the DSR losses are due to broken source routes; at 400 nodes and above, losses are mainly due to flooding-induced congestion. *Grid* does a better job than DSR over the whole range of numbers of nodes, especially for large networks.

Figure 5-8 shows the message overhead of the *Grid* and DSR protocols. Only protocol packets are included. In the case of *Grid*, these are HELLO, GLS update, and GLS query and reply packets. In the case of DSR, these are route request, reply, cached reply packets etc. DSR produces less protocol overhead for small networks, while *Grid* produces less overhead for large networks. At 400 nodes and above, DSR suffers from network congestion. Almost half of the route reply and cache reply messages are dropped due to

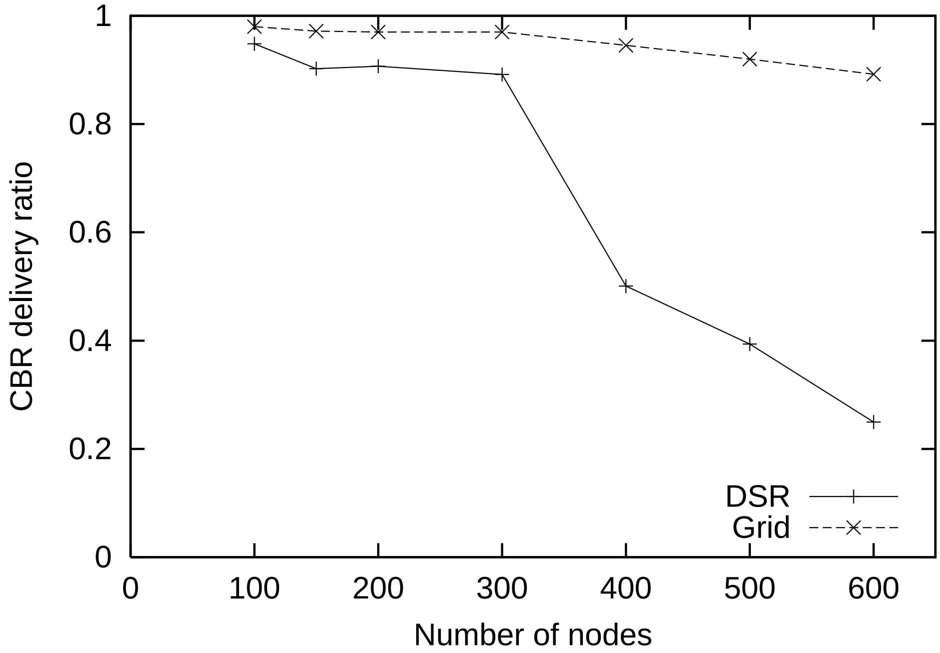

Figure 5-7: The fraction of data packets that are successfully delivered by DSR and *Grid* in networks of increasing numbers of nodes. The nodes move with a maximum speed of 10 m/s.

congestion which causes DSR to inject even more route requests into the network. Also, as the network grows larger and congestion builds up, the source route is more vulnerable to failure which will also induce DSR source nodes to send more route request packets. DSR's overhead drops at 600 nodes because it could not send much more packets in the presence of congestion. We present overhead in terms of packets rather than bytes because medium acquisition overhead dominates actual packet transmission in 802.11, particularly for the small packets used by *Grid*.

5.4 Summary

Our simulations show that *Grid* has great potential to scale to a large ad hoc network. In particular, GLS protocol message and storage overhead grow slowly with the size of the network. GLS copes with node movement reasonably well, although in a highly mobile network, we expect to see more GLS protocol packets due to frequent location updates.

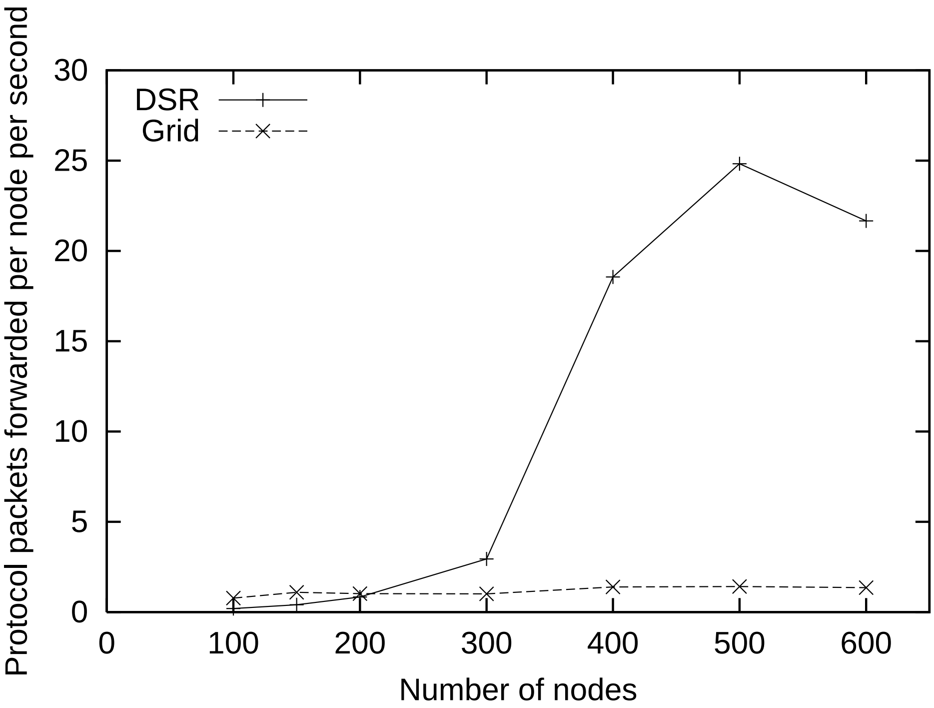

Figure 5-8: The number of all protocol overhead packets forwarded per node per second by DSR and *Grid* as a function of the total number of nodes. The nodes move with a maximum speed of 10 m/s.

Geographic forwarding is a good choice for mobile networks, as it is able to handle node movement and dynamically change the next hop much better than source routing as used in DSR. *Grid* outperforms DSR in large networks with greater than 300 nodes.

Chapter 6

Related Work

Many ad hoc routing protocols are designed with a flat hierarchy. Hence, there is a need in these systems for global operations that distribute entire network topology information or queries to all nodes in the network. Some, such as DSDV [17], are *pro-active*; they continuously maintain route entries for all destinations. Other techniques are *reactive*, and construct routes to destinations as they are required by flooding the network with query packets. This includes systems such as DSR [11], AODV [16], and TORA[15]. Broch et al. [5] and Johansson et al. [10] each provide overviews of these ad hoc routing techniques, along with comparative simulation results using small (30–50 node) simulations. They showed that pro-active routing protocols (DSR,AODV) deal with mobility well in small networks and caching is effective in reducing the frequency that route request messages are sent. In 0, 1 routing protocols such as DSDV could not cope with mobility well even in small networks because the routing table entries become stale easily when nodes move around.

For scalability concerns, a number of routing protocols have been proposed that actively maintain a hierarchy among mobile nodes to constrain the 0 of routing information. These systems also come with their own location service protocol in order to map each node's identifier with its changing hierarchical address used for routing.

MMWM [18] organizes nodes into a hierarchy of clusters and distributes link state information so that each node in the network knows how to route to sibling clusters of every ancestor cluster in addition to every node in the same lowest-level cluster. In MMWM, each node has a current hierarchical routing address which is a concatenation of the identifiers of the clusters that the node current resides in. Each cluster (of every size) has a designated location manager that keeps track of the next level of clusters the node belongs to currently. This protocol has been shown to work well in a simulation of battlefield battalions where groups of nodes move together and form natural clusters.

The Landmark system [19, 20] actively maintains a hierarchy to provide routing in a changing network. Nodes in a Landmark network have unique permanent IDs that are not directly useful for routing. Each node also has a changeable Landmark address, which consists of a list of IDs of Landmark nodes along the path from a well-known root to the node's current location. A Landmark address can be used directly for routing, since it is similar to a source route. The Landmark system provides a location service that maps IDs to current addresses. Each node **X** sends updates containing its current Landmark address to a node that acts as its address server, chosen by hashing **X**'s ID to produce a Landmark address A. If a node **Y** exists with that address, **Y** acts as **X**'s location server. Otherwise the node with Landmark address closest to A is used. Anyone looking for **X** can use the same algorithm to find **X**'s location server, which can be queried to find **X**'s current Landmark address.

Building a hierarchy among landmarks and nodes themselves is difficult in a mobile environment. Clustering approaches, such as MMWM, face problems due to inconsistent information at different nodes because of the asynchronous nature of the network. Consequently, the routing protocol is more complex and prone to error because of the need to be able to continue to function with inconsistent information. Moreover, the performance of such routing protocols depend heavily on the dynamic hierarchy being built. If there is no intrinsic hierarchy in the mobile network such as 0 in different units of the army, the protocol might have to initiate too many changes in hierarchy which are expensive operations. However, this combination of location servers and addresses that encode routing information is similar to the architecture described in this thesis. *Grid*, however, avoids building dynamic hierarchies, as they are vulnerable to the movement of nodes, especially changes near the top of the hierarchy.

More closely related to *Grid* are protocols that make use of geographic positions. Finn's

Cartesian routing [8] addresses each node with a geographic location as well as a unique identifier. Packets are routed by sending them to the neighbor closest to the packet's ultimate destination. Dead ends encountered in the forwarding are handled by scoped flooding. However, Finn gives no detailed explanation of how node locations are found or how mobility is handled.

More recent work on geographic approaches to routing includes the DREAM [2] and LAR [14] systems. LAR is designed to to enhance the performance of reactive routing protocols by using geographic information to constrain the scope of flooding. LAR does not maintain location information for each node explicitly. Rather, it relies on the first flooded route request message of the reactive routing protocol to learn the destination's geographic location and limit subsequent route request messages to be in a request zone consisting of the sender and its 1 1. LAR also does not explicitly use geographic information in making forwarding decisions. DREAM routes packets geographically, in a manner similar to Finn's Cartesian system. DREAM nodes 1 flood position updates over the whole network, allowing all the nodes in the network to build complete location databases. Because both LAR and DREAM involve global flooding, neither system seems suited to large networks.

Chapter 7

Conclusions

Wireless technology has the potential to dramatically simplify the deployment of data networks. For the most part this potential has not been fulfilled: most wireless networks use costly wired infrastructure for all but the final hop. Ad hoc networks can fulfill this potential because they are easy to deploy: they require no infrastructure and configure themselves automatically. But previous ad hoc techniques do not usually scale well to large networks.

We have presented a mobile ad hoc networking protocol with significantly better scaling properties than previous protocols. Although somewhat complicated to understand, our protocol is very simple to implement. In many ways the two facets of our system, geographic forwarding and the GLS, operate in fundamentally similar ways. Geographic forwarding moves packets along paths that bring them closer to the destination in physical space, only reasoning about nodes with nearby locations at each step along the path. GLS moves packets along paths that bring them closer to the destination in ID space, using only information about nodes with nearby IDs at each step along the path. Both mechanisms are scalable because they only need local information in their respective spaces.

Grid is an ad hoc routing system that promises to scale to large networks. However, it faces more challenges in order to be deployed. For example, what if nodes do not know their current location? Will geographic routing "holes" become much more common in real networks where obstacles and uneven radio ranges are present? We expect to address these practical issues as part of future work.

Bibliography

[1] USCG Navigation Center GPS page, January 2000. from `http://www.navcen.uscg.mil/gps/default.html`.

[2] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). In *Proc. ACM/IEEE MobiCom*, pages 76–84, October 1998.

[3] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with guaranteed delivery in *ad hoc* wireless networks. In *Proc. 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 48–55, August 1999.

[4] Josh Broch, David Johnson, and David Maltz. The Dynamic Source Routing protocol for mobile ad hoc networks. Internet draft (work in progress), Internet Engineering Task Force, October 1999. `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-03.txt`.

[5] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. ACM/IEEE MobiCom*, pages 85–97, October 1998.

[6] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. New York, New York, 1997. IEEE Std. 802.11–1997.

[7] Kevin Fall and Kannan Varadhan. *ns* notes and documentation. Technical report, UC Berkeley, LBL, USC/ISI, and Xerox PARC, November 1997. `http://www-mash.berkeley.edu/ns`.

[8] Gregory G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. ISI/RR-87-180, ISI, March 1987.

[9] CMU Monarch Group. CMU Monarch extensions to *ns*. `http://www.monarch.cs.cmu.edu/`.

[10] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proc. ACM/IEEE MobiCom*, pages 195–206, August 1999.

[11] David B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 158–163, December 1994.

[12] D. R. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. 29th ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[13] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. ACM/IEEE MobiCom*, August 1999.

[14] Young-Bae Ko and Vaidya Nitin H. Location-Aided Routing (LAR) in mobile ad hoc networks. In *Proc. ACM/IEEE MobiCom*, pages 66–75, October 1998.

[15] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. IEEE Infocom*, pages 1405–1413, April 1997.

[16] Charles Perkins, Elizabeth Royer, and Samir R. Das. Ad hoc On demand Distance Vecor (AODV) routing. Internet draft (work in progress), Internet Engineering Task Force, October 1999. from `http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-04.txt`.

[17] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proc. ACM SIGCOMM Conference (SIGCOMM '94)*, pages 234–244, August 1993.

[18] R. Ramanathan and M. Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications*, (4), 1998.

[19] Paul F. Tsuchiya. The Landmark hierarchy: A new hierarchy for routing in very large networks. In *Proc. ACM SIGCOMM Conference (SIGCOMM '88)*, pages 35–42, August 1988.

[20] Paul F. Tsuchiya. Landmark routing: Architecture, algorithms, and issues. MTR-87W00174, MITRE, May 1988.