A Scalable Location Service for Geographic Ad Hoc Routing

Jinyang Li   John Jannotti   Douglas S. J. De Couto   David R. Karger   Robert Morris
M.I.T. Laboratory for Computer Science
{jinyang, jj, decouto, karger, rtm}@lcs.mit.edu

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.

1. Introduction

This paper considers the problem of routing in large ad hoc networks of mobile hosts. Such networks are of interest because they do not require any prior investment in fixed infrastructure. Instead, the network nodes agree to relay each other's packets toward their ultimate destinations, and the nodes automatically form their own cooperative infrastructure. We describe a system, *Grid*, that combines a cooperative infrastructure with location information 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.

Jinyang Li and John Jannotti are supported by DARPA contract N66001-99-2-8917. David Karger is supported by NSF contract CCR-9624239, an Alfred P. Sloane Foundation Fellowship, and a David and Lucille Packard Foundations Fellowship.

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 networks to meet increasing bandwidth requirements 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. Finally, existing services may not provide adequate service, or may be too expensive.

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 proactively distribute network topology information among the nodes, and each node pre-computes routes through that topology using relatively inexpensive algorithms. Second, fixed networks 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 are likely to be close in the network topology; that is, they will be connected by a small number of radio hops.

Grid uses geographical forwarding to take advantage of the similarity between physical and network proximity. A source must know the geographical positions of any destination to which it wishes to send, and must label packets for that destination with its position. 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 be useful in a larger context, a system based on geographic forwarding must also provide a mechanism for sources to learn the 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. It should also be scalable in the following senses:

1. No node should be a bottleneck—the work of maintaining the location service should be spread evenly over the nodes.
2. The failure of a node should not affect the reachability of many other nodes.
3. Queries for the locations of nearby hosts should be satisfied with correspondingly local communication. 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.

The Grid location service (GLS) presented in this paper satisfies all of these requirements.

The rest of the paper describes the design and simulated performance of Grid. Section 2 reviews existing work in scalable ad hoc networking. Section 3 describes the characteristics of geographic forwarding. Section 4 describes Grid's distributed location service algorithm. Section 5 describes our implementation of geographic forwarding and the GLS in detail. Section 6 analyzes Grid's routing performance and scalability using simulations. Section 7 suggests areas for future improvements. Section 8 summarizes the paper's contributions.

2. Related Work

Most existing ad hoc routing systems distribute either topology information or queries to all nodes in the network. Some, such as DSDV [16], are *proactive*; they continuously maintain route entries for all destinations. Other techniques are *reactive*, and construct routes to destinations as they are required. This includes systems such as DSR [10], AODV [15], and TORA[14]. Broch et al. [4] and Johansson et al. [9] each provide overviews of these ad hoc routing techniques, along with comparative measurements using small (30–50 node) simulations. Grid's main contribution compared to these works is increased scalability.

More closely related to Grid are protocols that use geographic positions. Finn's Cartesian routing [7] 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 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 [13] systems. Both systems route packets geographically, in a manner similar to Finn's Cartesian system. They differ in how a node acquires the geographic position of a destination. DREAM nodes proactively flood position updates over the whole network, allowing other nodes to build complete position databases. LAR nodes reactively flood position queries over the entire network when they wish to find the position of a destination. Because they both involve global flooding, neither system seems suited to large networks.

The Landmark system [17, 18] 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 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. This combination of location servers and addresses that encode routing information is similar to the architecture described in this paper. Grid, however, avoids building hierarchies, as they are vulnerable to the movement of nodes near the top of the hierarchy.

3. Geographic Forwarding

We use a simple scheme for geographic forwarding that is similar to Cartesian routing [7]. 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. When node needs to forward a packet toward location *P*, the node consults its neighbor table and chooses the neighbor closest to *P*. It then forwards the packet to that neighbor, which itself applies the same forwarding algorithm. The packet stops when it reaches the destination.

A packet may also reach a node that does not know about any nodes closer than itself to the ultimate destination. This dead-end indicates that there is a "hole" in the geographic distribution of nodes. In that case, the implementation described in this paper gives up and sends an error message to the packet's source node.

Recovering from dead-ends is possible using the same neighbor position table used in geographic forwarding. Karp and Kung propose GPSR [12], a geographic routing system that uses a planar subgraph of the wireless network's graph to route around 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. 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.

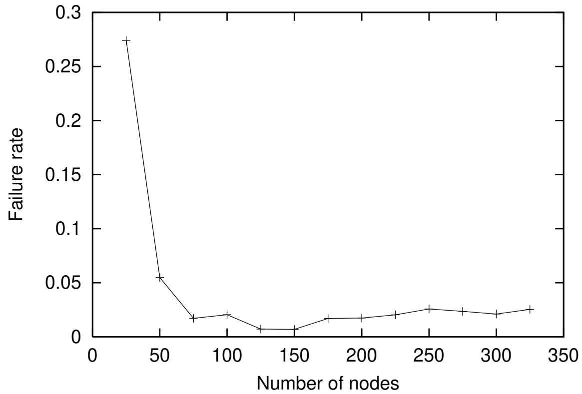

Figure 1: Fraction of data packets unable to be delivered using geographic forwarding with a perfect location service, as a function of node density. The simulation area is 1 km^2.

3.1 Effect of Density

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* [6] network simulator. The simulated nodes have 2 Megabit per second IEEE 802.11 radios [5] 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 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 random; 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.

Figure 1 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 $141 = 1000/\sqrt{50}$ 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.

4. 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. A trivial location service might consist of a statically positioned location server. Nodes would periodically update this server (using geographic forwarding to the 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 can not allow multiple network partitions to each function normally in their own partition; and nodes near to each other gain no advantages—they must contact a potentially distant location server in order to communicate locally.

We introduce a distributed location service (GLS) that is designed 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 "out of the way" of a straight line trip from the node performing the lookup to the node being looked up.

GLS is based on the idea that a node maintains its current location in a number of *location servers* distributed throughout the network. 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 are proportional to data path lengths.

In order to spread uniformly the work of acting as location servers, GLS avoids techniques such as leader election or 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 [11]. 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 target. 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 there is a random distribution of node IDs across the network. GLS ensures that nodes are allocated unique, random IDs 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**.

4.1 Selecting and Querying Location Servers

GLS provides for distributed location lookups by replicating the knowledge of a node's current location at a small subset of the net-

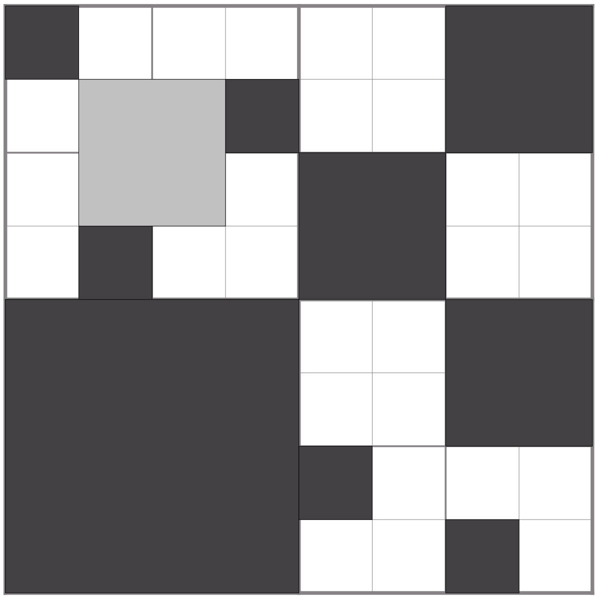

Figure 2: A piece of the global partitioning of the world. A few example squares of various orders are shown with dark shading. The lightly shaded square is shown as an example of a 2x2 square which is *not* an order-2 square because of its location. An order-n square's lower left corner's coordinates must be of the form $(a2^{n-1}, b2^{n-1})$ for integers a,b.

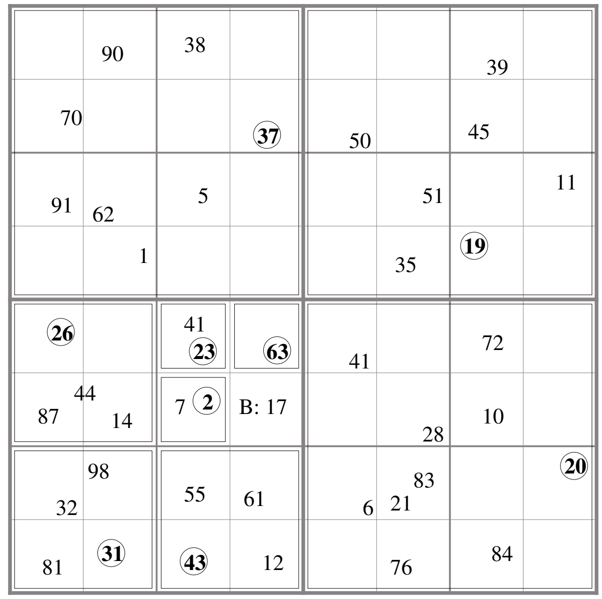

Figure 3: The inset squares are regions in which B will seek a location server. The nodes that become B's location servers are circled and shown in bold.

work's nodes. This set of nodes is referred to as the node's location servers. A node **A** hoping to contact node **B** can query one of a number of other nodes that know **B**'s location. Of course, **A** must be able to contact the nodes that know **B**'s location. This means that **A**'s search for **B**'s location servers and **B**'s original recruitment of location servers ought to lead to the same servers. When **B** recruits location servers it uses the same information that **A** will have when searching for **B**'s location servers: **B**'s name and certain information that all nodes have at startup.

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 2. The smallest square is referred to as an order-1 square. Four order-1 squares make up an order-2 square, and so on. It is important that not every square made up of four order-n squares is also an order-$(n+1)$ square. Rather, to avoid overlap, a particular order-n square is part of only one order-$(n+1)$ square, not four. This maintains an important invariant: a node is located in exactly one square of each size. 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 instead.

Consider how **B** determines which nodes to update with its changing location, using its ID and the predetermined grid hierarchy. **B** knows that other nodes will want to locate 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**. The ID space is considered to be circular, 2 is closer to 17 than 7 is to 17.

If we consider the tree corresponding to the grid decomposition, 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). A node chooses three location servers for each level of the grid hierarchy. For example, in the figure, **B** recruits three servers in order-1 squares, three servers in order-2 squares, and three servers in order-3 squares. In each of the three order-1 squares that, along with **B**'s own order-1 square, make up an order-2 square, **B** chooses the node closest to itself in ID space as a server. The same location server selection process occurs in higher order squares. In the three order-2 squares that combine with **B**'s order-2 square to make an order-3 square, **B** selects 26, 31, and 43 as location servers.

Figure 4 shows the state of a Grid network once all nodes have provided their coordinates to the nodes that will act as their location servers. 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** has location information. That node forwards the query in the same way, and so on. Eventually, the query will reach a location server of **B** which will foward the query to **B** itself. Since the query contains **A**'s location, **B** can respond directly using geographic forwarding. The location query is forwarded all the way to **B** so that **B** can respond with its latest location.

For illustrative purposes we have ignored an important bootstrapping issue. We have assumed that nodes select their location servers appropriately and send their coordinates to them. This appears to assume that a node can scan an entire square (of arbitrary size) and choose the appropriate node to act as its server. In fact, nodes route update packets to their location servers without knowing their identities. Assume that a node **B** wishes to recruit a location server in some order-n square. **B** sends a packet, using geographic forward-

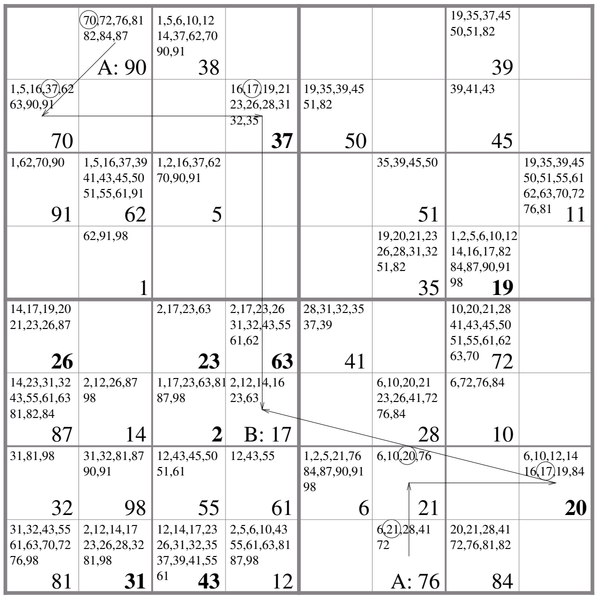

Figure 4: An entire network's location server organization. Each node is shown with the list of nodes for which it has up to date location information; B's location servers are shown in bold. Two possible queries by A for B's location are shown.

ing, 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 below, the update will arrive at the least node greater than **B** before leaving the order-n square containing **L**. This is exactly the appropriate destination 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 an order-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, order-1 squares would exchange information using the local routing protocol, then nodes could recruit their order-2 location servers, then order-3, etc. Once the order-n location servers are operating, there is sufficient routing capability to set up the order-$(n+1)$ location servers.

4.2 Efficiency Analysis

When nodes are not moving, the number of steps taken by a location query from **A** to **B** is no more than the order of the smallest square in which **A** and **B** are colocated. A location query *step* is distinct from a single *hop* in the geographic forwarding layer; indeed, each location query step is likely to require several geographic forwarding hops. In Figure 4, the entire diagram is an order-4 square. Therefore all queries can be performed in no more than four location query steps.

At each step, a query makes its way to the best (closest in ID space to the destination) node at successively higher levels in the grid hierarchy. At the start, the query is forwarded to the best node in the local order-1 square using the local routing protocol. From this point on, each step moves the query to the best node in the next larger containing square; when that next larger square contains the destination node, the best 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 best node 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 4. Our abbreviated topology has no more than one node per square, so the query trivially begins at the best node, itself, in its order-1 square. The query moves to the best node (21) in **A**'s order-2 square, because 76 happens to know the positions for all the nodes in its order-2 square. This is an artifact of our sparse layout, so the next step tells the important story: why 21 knows the location of the best node in the next higher order square.

Recall that 21 is the best node in its order-2 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 order-3 square, but not in 21's order-2 square. Recall that **X** had to choose a location server in node 21's order-2 square. If **X**'s ID is between 17 and 21 then **X** *must* have chosen node 21 as a location server since there are no better nodes in node 21's order-2 square. Thus, node 21 knows about *all* nodes in its order-3 square that lie between 17 and itself, including the minimum such node. In this case, that node is 20. At the next step, node 20 must know about all nodes in the order-4 square between 17 and itself. Since nodes 20 and 17 share the same order-4 square (the entire 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 the best node in its order-2 square to the best node in its order-3 square. One may wonder, however, why node 21 does not know about some other node whose ID is between 17 and 20, and 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 node outside of the lower right quadrant of figure 4, node 21 is guaranteed *not* to be the best choice for location server; node 20 will always be preferable. In addition, because every location query is labelled with its source, intermediate query steps know what level of the hierarchy the query is currently in, and can refrain from sending queries too far away.

Having built an intuition, we now give an inductive proof that a query needs no more than n location query steps to reach its destination when the source and destination are colocated in an order-n square. Furthermore, the query never leaves the order-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 or fewer location query steps, a query reaches the node with the lowest ID (i.e closest to 0) in the order-n square containing the source. Since the destination is node 0, when the query reaches the order-n square that contains both the source and the destination nodes, it must reach the destination.

Base case (order-1 square): The query begins at a node **X**. Node **X** may or may not be the node with the lowest ID in its order-1 square. If so, the query trivially reaches the lowest node in the order-1 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 order-1 square, **Y**, through the local routing protocol. Node **X** will not know of any other nodes with IDs lower than **Y**. Any such node would not have selected **X** as a location server as **Y** would always have been 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.

Inductive step (order-$(n+1)$ square): We claim that if a query is at the node **X** with the lowest ID in its order-n square, then **X** will route the query to the node **Y** with the lowest ID in its order-$(n+1)$ square with one or zero location query steps. If **X** has the lowest node ID in the order-$(n+1)$ square, then our claim is trivially true. If not, **X** will know the coordinates of **Y** and *will not* know the coordinates of any node lower than **Y** outside the order-$(n+1)$ square. Node **X** will know the coordinates of **Y** because **Y** will have selected **X** as a location server. Node **Y** must have selected a location server in **X**'s order-n square because **Y**'s order-n square is a part of the same order-$(n+1)$ square as **X**'s. Node **Y** must have selected **X** because **X** is the lowest node in its square that is greater than **Y**. Node **X** will not know the location of any node lower than **Y** outside of its order-$(n+1)$ square because when any such node sought a location server in **X**'s order-$(n+1)$ 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 remember however, that this proof applies only to a static network. Additional techniques, described in Section 5, help Grid to deal with the problems cereated by mobility. These sections describe 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.

5. Implementation

This section describes the details of the geographic forwarding and GLS protocols.

5.1 Geographic Forwarding

The geographic forwarding layer uses a two hop distance vector protocol. This helps alleviate holes in the topology and ensures that each node knows the location of all nodes in its order-1 square. Each node maintains a table of immediate neighbors as well as each neighbor's 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 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. The routing table is also updated every time a node receives a packet, using the packet's last hop information.

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 recorded speed. If it would likely still be in range, the entry may still be used for forwarding, but it is not 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 congestion 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 path is chosen.

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

Figure 5: HELLO packet fields.

To select a next hop, nodes first choose a set of nodes from all nodes in their neighbor table. This set consists of the best nodes to move the packet to, as defined by the shortest distance to the destination from the candidate nodes. All nodes whose distances to the destination are nearly equal are considered in this set. 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 if it was a single-hop neighbor).

5.2 Updating Location Information

GLS maintains two tables in each node. The location *table* holds the node's portion of the distributed location database; 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. 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 order-1 square, the node does not need to send normal GLS updates within its order-1 square.

As a node moves, it must update its location servers. Nodes avoid generating excessive amounts of update traffic by linking their location update rates to their distance traveled. A node updates its order-2 location servers every time it moves a particular threshold distance d since sending the last update; the node updates its order-3 servers after each movement of $2d$. In general, a node updates its order-i servers after each movement of $2^{i-2}d$. This means that a node sends out updates at a rate proportional to its speed and that updates are sent to distant servers less often than to local servers. In addition, nodes send location updates at a low rate even when stationary.

Location update packets (see Figure 6) 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 obtained from different nodes for the same destination can be com-

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

Figure 6: GLS update packet fields.

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 7: GLS query packet fields.

pared. GPS receivers can provide every node in the network with closely synchronized time.

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.

Nodes piggyback their location information on data packets, so that two nodes who are communicating always know how to reach each other. In the case of one-way communication, nodes also periodically send their position information directly to nodes who are sending them data.

5.3 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 packet to **D**'s recorded location using geographic forwarding. Otherwise, **S** initiates a location query for **D** using the GLS. GLS will eventually deliver the query packet (Figure 7) to **D**, which will geographically route a response to **S** that includes **D**'s current location.

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

5.4 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 occurs 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 order-1 square s_1 to the order-1 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 order-1 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.

6. Performance Analysis

This section presents simulation results for GLS 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 work: the number of location database entries each node must store, and the number of protocol packets each node must originate or forward in order to route a given workload. The simulations show that these costs 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 resent. 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.

6.1 Simulation Scenario

The simulations use CMU's wireless extensions [8] for the *ns* [6] 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

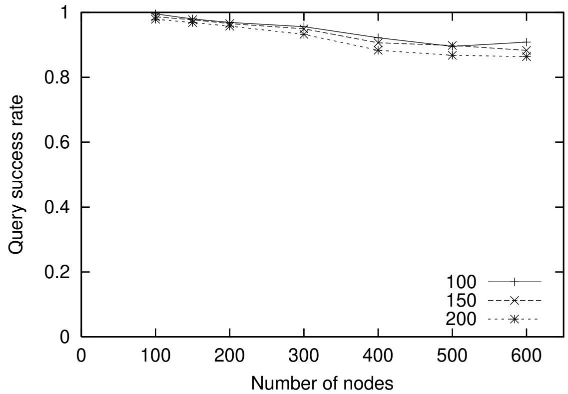

Figure 8: GLS query success rate as a function of the total number of nodes. The nodes move at speeds up to 10 m/s (about 22 miles per hour). Each line corresponds to a different movement update threshold.

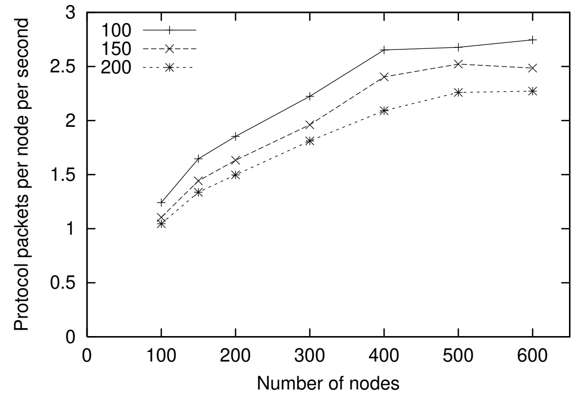

Figure 9: Average number of Grid protocol packets forwarded and originated per second by each node as a function of the total number of nodes. Nodes move at speeds up to 10 m/s.

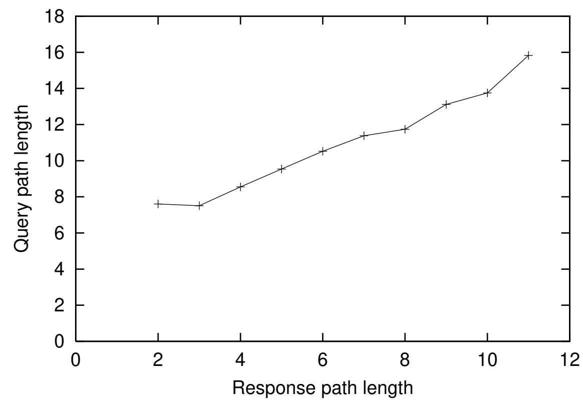

Figure 10: Average query path length (in hops) as a function of the query reply path length, for 300 nodes moving up to 10 m/s.

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 order-1 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 order-5 in a square universe 2900m on a side.

Each node moves using a "random waypoint" model [4]. 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.

6.2 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 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 8 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 9 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 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 9 is generated from the same simulations that produced Figure 8. The graph shows that Grid imposes a modest protocol traffic load as the network size grows.

Figure 10 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

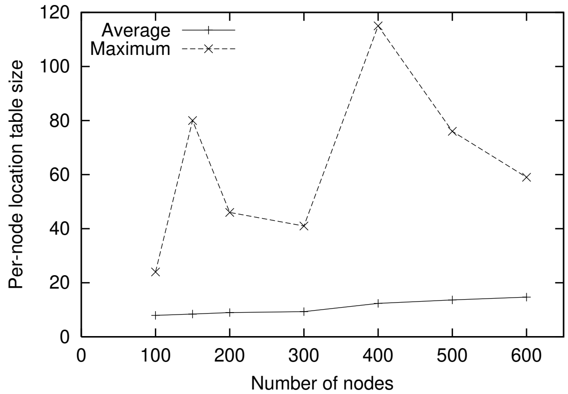

Figure 11: Average and maximum per-node location database size (number of entries) as a function of the total number of nodes. The nodes move at speeds up to 10 m/s.

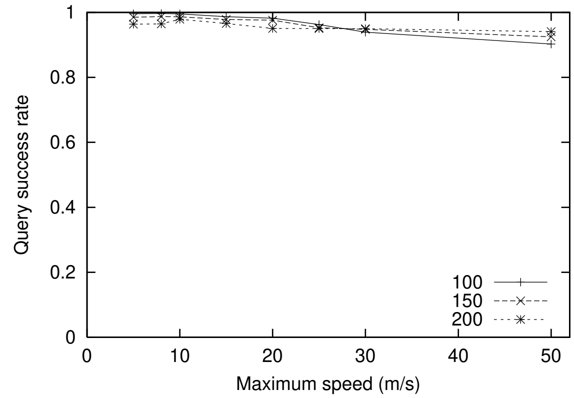

Figure 12: GLS query success rate as a function of maximum node speed in a network of 100 nodes. 50 m/s is about 110 mph.

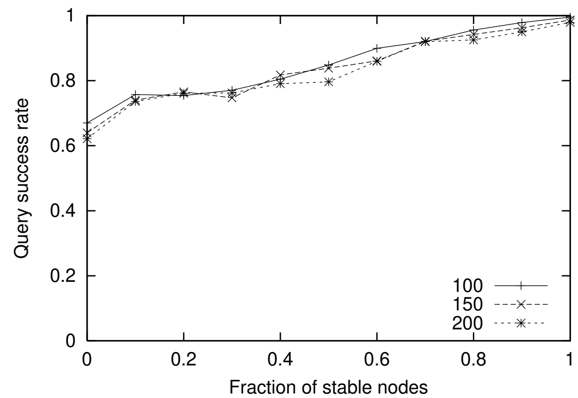

Figure 13: The effect of turning off nodes on the query success rate. The X axis indicates the fraction of nodes that are always on; the remaining nodes cycle on and off for random periods up to 120 and 60 seconds, respectively. The simulations all involve 100 nodes moving at speeds up to 10 m/s.

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 with 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 11 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 12 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 13 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 was manually turned off, it would be appropriate to first 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 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.

6.3 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

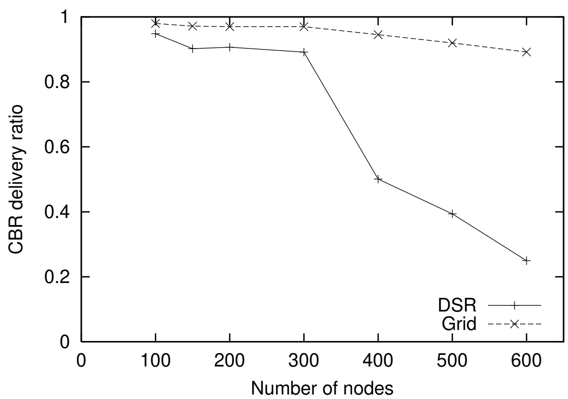

Figure 14: The fraction of data packets that are successfully delivered in simulations for increasing numbers of nodes. The nodes move with a maximum speed of 10 m/s.

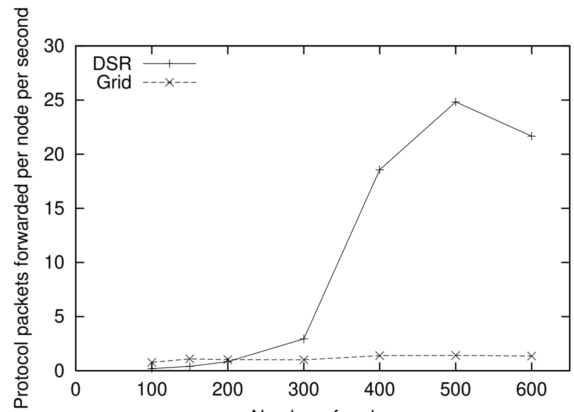

Figure 15: The number of all protocol packets forwarded per node per second as a function of the total number of nodes. No data packets are included. The nodes move with a maximum speed of 10 m/s.

purposes of comparison we include results for the DSR [10] protocol. This may not be a fair comparison since DSR is optimized for relatively small networks [3].

Figure 14 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 15 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 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.

7. Future Work

One area of the GLS protocol that could be improved is the handling of node mobility. Accurate movement models may allow us to integrate movement prediction into the GLS protocol. Our current system makes little effort to predict the movement of nodes over long time periods because our movement model is randomized, but in the real world a node may not need to update a location server as often if its velocity is constant or predictable.

Currently the GLS protocol makes little effort to proactively correct out-of-date information when, for instance, a node crosses a grid boundary line. Proactive updates may reduce the incidence of query failures. However, the tradeoff is obvious—care must be taken not to consume too much bandwidth with the updates. An alternate strategy to address the same problem is to place less trust in locations obtained from distant location servers. Rather than trust a distant location server to pinpoint the order-1 square in which a node is located, a query could be moved to, for instance, the surrounding order-3 square. There the query can be restarted with the fresher information available in that square

Another potential area of improvement is adapting to node density. If an order-1 square becomes too crowded, each node will get less bandwidth from the shared radio spectrum, and each node will have to work harder to keep its neighbor table up to date. Radios with variable power levels would help alleviate this problem by changing the effective density of nodes within radio range. In addition, each square in the GLS may make a local decision about how finely to sub-divide itself; distant areas need not agree on the size of the order-1 square.

Finally, as we noted earlier, the choice of a grid based system is somewhat arbitrary. In fact, certain partitioning schemes offer the possibility of better scaling. The number of location servers that a node must recruit is equal to the number of neighbors per level in the geographic hierarchy multiplied by the number of levels in the hierarchy. For a grid based system, this means that a node must maintain $3 \log_4 n$ servers in a network that is n times the size of the coverage area of a single radio. It is possible, however, to split the world in half at each level, rather than in fourths, by using rectangles with an aspect ratio of $1/\sqrt{2}$. At successive levels, each such rectangle may be divided into two such rectangles. This leads to a network in which nodes must recruit only $\log_2 n$ location servers, or $2/3$ the number of servers needed in a grid based approach.

8. 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.

9. Acknowledgments

We would like to thank Frans Kaashoek for his valuable input which improved the paper. We would also like to thank Yan Zhang for her discussions and help with running simulations, and Brad Karp for reading a draft.

10. References

[1] USCG Navigation Center GPS page, January 2000. `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] 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`.

[4] 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.

[5] 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.

[6] 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`.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. ACM/IEEE MobiCom*, August 2000.

[13] 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.

[14] 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.

[15] 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. `http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-04.txt`.

[16] 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.

[17] 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.

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