Load Balancing the GRID Ad-Hoc Routing Protocol

by

Rebecca S Bloom

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

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author .....
Department of Electrical Engineering and Computer Science
August 22, 2003

Certified by .....
Dina Katabi
Assistant Professor
Thesis Supervisor

Certified by .....
David Karger
Associate Professor
Thesis Supervisor

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

Load Balancing the GRID Ad-Hoc Routing Protocol

by

Rebecca S Bloom

Submitted to the Department of Electrical Engineering and Computer Science
on August 22, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Grid is a mobile ad hoc routing system with significantly better scaling properties than previously designed protocols in networks with a uniform distribution of nodes. It achieves scalability by building a distributed location database in which memory requirements are apportioned fairly to all nodes in the network. In networks of spatially non-uniform node distribution, however, a small fraction of the nodes need to store a disproportionate amount of location information, posing a significant problem for nodes with limited memory such as small handheld devices. We propose a new location service to improve the scalability of Grid while preserving its fundamental design. Our new service, GLS2, ensures protocol correctness despite limited memory constraints. To preserve high success rate and improve query path efficiency, GLS2 applies a new location query algorithm which allows nodes to drop location information if necessary. Simulated tests demonstrate that GLS2's efficiency and correctness are preserved in situations of limited memory as well as those of extremely uneven node distributions while still achieving proper load-balancing.

Thesis Supervisor: Dina Katabi
Title: Assistant Professor

Thesis Supervisor: David Karger
Title: Associate Professor

Acknowledgments

I am indebted to many people at MIT for their support during my research. First and foremost, I am very grateful to Dina Katabi, my thesis adviser at the Laboratory for Computer Science, who gave me the incredible opportunity to work for her. Her guidance and support as well as our regular brainstorming sessions were essential in completing this work. I would like to thank David Karger for providing crucial feedback and new ideas. Indeed, it was due to his initial inspiration which enabled me to write this thesis. I must also thank Sophia Yuditzkaya for being such a wonderful partner for our team project in 6.829. Her diligence and patience in the early days of this project were invaluable. A special thank you to Jinyang Li, the primary author of the Grid paper and original Grid simulation code. In addition to introducing me to her code, she was always there to offer help and suggestions. Thomer Gil and Chuck Blake, thank you for reading a draft of this paper.

To my husband Elazar, words cannot express how I feel about you. You complete me. Thank you for an amazing year.

And to the rest of my family. Mom, Dad, Sarah, David, and Rachel, you never stopped believing in me. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In an increasingly mobile world, where pervasive computing is the next great paradigm, ad hoc wireless networking is of immense value because it needs no prior deployed network infrastructure. By routing packets for each other, nodes that participate in an ad hoc network spontaneously form their own cooperative networking infrastructure. Such flexibility eliminates the need for the administrative, temporal, and economic costs that characterize the construction of traditional wired networks.

Among the existing proposals for ad hoc routing protocols, Grid [8] holds great promise to scale to a large number of ad hoc nodes. Grid uses geographic forwarding to route packets to destination nodes using only local neighbor information. A scalable location service, GLS, is built among the ad hoc nodes to share geographic information of other nodes. [8] shows GLS is robust and load balanced in networks with a uniform distribution of nodes. However, in scenarios of uneven node distribution, the scalability assumptions made in the Grid paper prove inaccurate. GLS might cause some nodes to store an immense amount of location information to ensure protocol correctness. This is a tremendous concern in any system which memory is a constraint such as a network of small hand-held devices.

We therefore designed a protocol to improve the scalability of the Grid location service while preserving its fundamental design. To deal with a constraint on the amount of location information stored at each node, our new service functions correctly despite missing location data. To do this we employed a new algorithm that routes *location queries*, packets sent into the network specifically to find the location of certain nodes, to their proper destinations even in cases that location information does not exist. Through analysis, we have proved that in static environments, our protocol routes packets to their destinations efficiently and correctly.

Experiments using the *ns* simulator confirm that our new service routes a significant percentage of queries to their destinations while bounding the maximum amount of location information that a

node must retain. Additionally, the majority of failures encountered were a result of the underlying forwarding layer, not a result of our new algorithm.

The rest of this work is organized as follows. Chapter 2 reviews Grid's location service as well as related work. It also motivates our work by carefully describing the problem. Chapter 3 discusses our new location query algorithm. Chapter 4 presents a proof of correctness and termination as well as theoretical analysis of the protocol. Chapter 5 describes the simulation environment. We proceed to evaluate our simulation results in Chapter 6. Finally, Chapter 7 presents the conclusions.

Chapter 2

Background & Motivation

2.1 Grid and the Grid Location Service

We start this chapter by providing a few definitions that we use throughout this dissertation.

Table 2.1: Definition of Terms Used in Grid.

Term	Definition
Grid	A service which combines a geographical forwarding with location information to implement routing in a large ad hoc network.
GLS	Grid's Location Service, a distributed protocol which tracks the locations of mobile nodes.
Location Server for node X	A node whose location table contains an entry for node X (i.e., it knows the geographic location of X).
Location Table	Table mapping Node IDs to their geographic location.
Location Query	Packet sent into the network to retrieve the geographic location of the queried node.
Location Update	Packet sent by a node to its location server informing the server of the node's current location.
Sibling Grid	A grid colocated in the same level of the hierarchy as this grid.
CND_A	*Closest ID Node to the Destination of which node A knows.* The node with the least ID greater than or equal to the Destination's ID of which A knows.
Smallest Grid	An order-0 grid. It is defined to be the size of the coverage area of a single radio-range in which a node can announce itself.

2.1.1 System Overview

Among the existing proposals for ad hoc routing protocols, the Grid system combined with an underlying geographical forwarding layer is a desirable choice because of its robustness, scalability, and simplicity. Grid achieves robustness by using a Geographical Location Service (GLS) which replicates forwarding information in geographically diverse locations. GLS partitions the universe into a quadtree, a hierarchical grid which can be visualized as a succession of subdivisions: first, the universe is subdivided into four quarters, then each quarter is further subdivided into four quarters, and so on. The grid lines are static and known to all nodes in the network in advance. At each level of the hierarchy, a node "recruits" three location servers, one location server in each of the three sibling grids that do not enclose that node as illustrated in Figure 2-1. Grid requires no extra infrastructure to execute location service as every node in the network is itself a potential location server for other nodes; each has a location table in which it stores a mapping of node IDs to current locations.

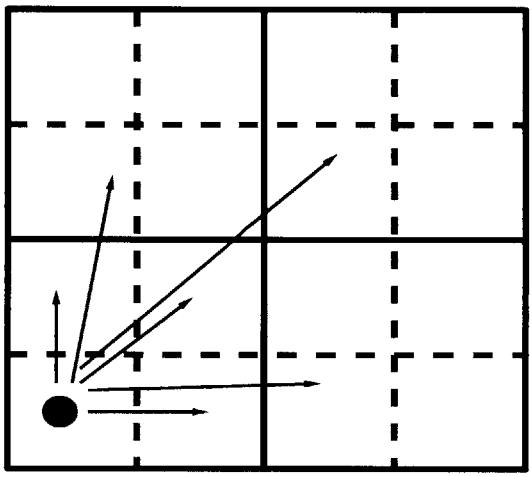


Figure 2-1: A node sending out its location information to location servers in each of its sibling squares at each level. The location information will be stored in the least node greater than itself within the specified square.

2.1.2 Geographical Forwarding

The underlying routing layer in Grid is that of geographic forwarding which capitalizes on the topological assumption for radio-based ad hoc networks: nodes that are physically close to each other are likely to be close in network topology. A node determines its own location using a system such as GPS [10] and then periodically broadcast this information to its neighbors. To send a packet to a specific location, a node forwards it to the neighbor closest in physical distance to the destination.

2.1.3 Assigning Location Servers

We now review the process of assigning location servers. The protocol assumes that each node has a unique identification number (ID) as a location server is defined to be the successor node with the least ID greater than or equal to the recruiting node's ID in each appropriate hierarchical subdivision. To recruit a location server in one of its order-n sibling squares, node **B** sends out a location update packet via geographic forwarding to any node in this square. After the packet arrives in this updated subdivision, a forwarding process based on node ID commences. The node in the subdivision forwards the packet to the closest ID node to **B** of which it knows in the order-n square. This process continues until the packet arrives at a node which knows of no node smaller than itself which is greater than **B** in the updated subdivision. This node thus becomes the location server for **B** and stores its location information in its table. Note that **B** does not need to know its location servers' IDs to recruit them. All that is required for proper routing of update packets is that the nodes' order-n location servers be recruited before their order-(n+1) location servers.

2.1.4 Location Queries

When one node **A** wishes to contact another node **B**, it performs a query for **B**'s location using a similar protocol to that of sending location updates. **A** sends a query packet to the closest ID node to the destination (CND_A) for which it has information in its location table. Let's call this node **C**. When **C** receives the packet, it forwards the query to the closest ID node to **B**, (CND_C), of which it knows. The query continues to be forwarded to nodes that are monotonically closer to the destination in ID space than the previous hop until eventually it reaches the destination. The destination then responds by sending the source node its location via geographic forwarding.

2.1.5 Efficiency

In a static network, the upper bound on the number of steps taken by a location query from **A** to **B** is the order of the smallest square in which **A** and **B** are colocated. This is because at each step of the query, the packet is sent to the closest ID node to the destination at successively higher levels in the grid hierarchy. The query ends when the next larger square contains the destination node. The CND in this square is **B** itself. For a detailed proof, see the original Grid paper [8]. This behavior also puts a bound on the geographic region in which the query will propagate. The query will only proceed to search squares of increasing orders that contain the source. Thus, the query will always find the destination colocated in **A**'s order-n square without traveling to **A**'s order-(n+1) square.

2.1.6 Scalability

GLS is is completely distributed, statistically spreading out routing responsibility uniformly among all nodes in the network. This distribution property is achieved through a technique similar to *Consistent Hashing* [6]. Like the hash function used to build a distinct hierarchy for each page, Grid uses a distinct location service hierarchy for each node to avoid making any node a bottleneck of the distributed location database. In contrast, alternative ad hoc protocols often use leader election and hierarchy to designate certain "unlucky" nodes for the role of pseudo-centralized routing and forwarding [12]. In an ad hoc network where no nodes have significantly more bandwidth and storage resources than others, a load balanced protocol like GLS is more scalable.

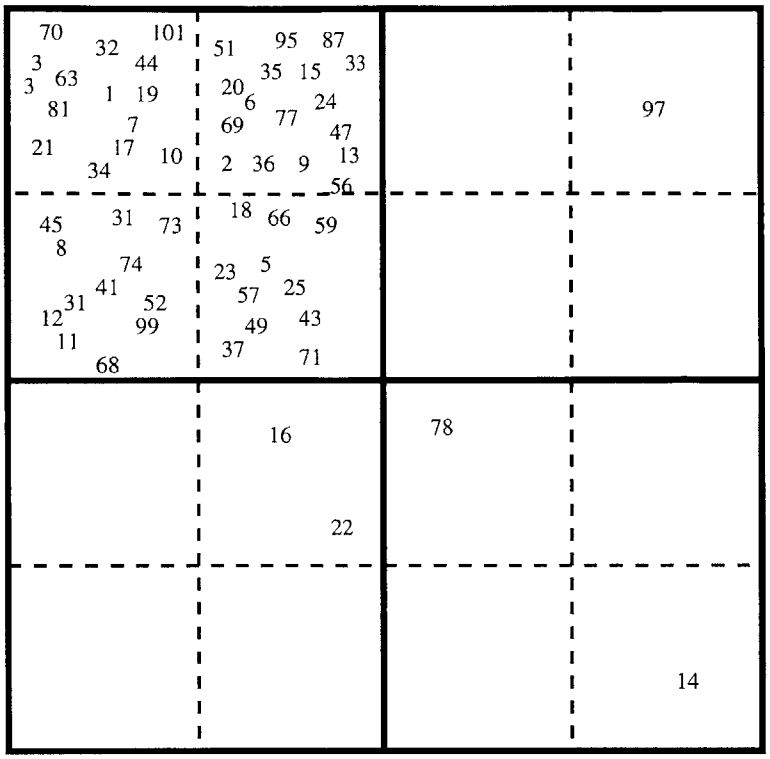
Additionally, GLS achieves scalability by minimizing the overhead required to maintain the system. The hierarchical structure of the quadtree guarantees that the number of a node's location servers decreases logarithmically with increasing distance to the node [5]. Thus, a node sends the majority of its location update packets to its closer location servers which reduces bandwidth consumption. This heirarchical designation of location servers ensures robustness; a node will have a number of location servers in close proximity.

2.2 Problem

GLS has been proposed, implemented, and evaluated in the context of a uniform distribution of nodes across the universe [8]. Simulation results show that the Grid protocol has significantly better scaling properties than previously proposed ad hoc designs. However, GLS was not analyzed in networks of non-uniform node distributions. In these scenarios a fundamental claim of GLS no longer holds true: The per-node storage requirement will not necessarily grow as a small function of the total number of nodes. Indeed, to ensure protocol correctness, some nodes might be required to store the location information of almost every node in the network. This potentially immense storage requirement adversely affects GLS's scalability.

Consider how GLS performs in environments with extremely uneven distribution of nodes, such as that shown in Figure 2-2. Appendix A gives the mapping of each node in the sparsely populated region to the nodes for which it will be a location server. As observed, a small number of nodes are assigned to be location servers for many nodes in the densely packed area of the universe. In this scenario, where nodes are not evenly distributed throughout the grid hierarchy, GLS places tremendous memory requirements on the location servers in the sparse region of the world. This unfair storage burden defeats the purpose of the *Consistent Hashing* technique that was designed to statistically spread the load equally across all nodes in the network. Without proper load balancing, scalability cannot be easily achieved.

We therefore propose a new protocol, GLS2, to improve the scalability of Grid in situations of



Extremely uneven node distribution

Figure 2-2: Nodes 14, 16, 22, 78, and 97 are the nodes in the sparse area which are forced to store location information for the majority of the other nodes.

uneven node distribution while preserving the high efficiency and correctness of the original GLS algorithm.

Our new protocol:

- Improves the scalability of GLS in networks of uneven node distribution.
- Improves the practicality of GLS because it allows each node to independently choose its buffer size based on its own resources. It does not assume that nodes have unlimitted memory to invest in building location tables.
- Performs at least as well as GLS in scenarios of uniform distribution. We measured performance based on the number of successfully routed queries.
- Preserves the original design of the original GLS algorithm.
- Preserves the efficiency of GLS. As with the original GLS, a query should never travel to a distant grid in order to find a destination located in a grid closer to the source.

2.2.1 Similar Problems

Note that although we emphasize the improvement to scalability of networks with non-uniform node distributions, our protocol can correctly and efficiently route packets in other similar scenarios where location information might be constrained. Take, for example, networks containing nodes of disparate storage capacities. In this situation, Grid cannot ensure correctness unless every node has a large enough buffer to store the location information requested of it. Our new protocol ensures that packets be routed to their destinations when there is a per-node bound on table size.

Another scenario which GLS2 improves is networks with frequent node joins. In these scenarios, a node might not have had time to receive all of its proper location information before being queried. Our new algorithm will still route the queries to their proper destinations. This could be a tremendous benefit for any system attempting to conserve battery power. To save energy, nodes might enter a *power saving mode* in which they do not receive any update packets. Nodes could frequently fully re-join the network and then leave again without concern as to a breakdown of correctness.

2.3 Related Work

Prakash, Haas, and Singhal designed a load-balanced location management scheme that takes into account non-uniform node distributions [11]. In their protocol, the set of location servers for a particular node changes over time. Their simulation experiments demonstrate that such dynamism prevents situations of heavy load on some location servers when mobile nodes are not uniformly distributed in space. However, their solution requires fixed infrastructures, relying on designated base stations to store location information and fixed wirelines connecting those stations. GLS2 is a completely ad hoc system requiring no fixed infrastructure.

Kasemann and Hartenstein [7] studied the impact of node density on the performance of GLS, but they vary overall node density while keeping the nodes uniformly distributed throughout the universe. They do not consider a universe that contains a variety of node densities as does our protocol.

Amis and Prakash propose a heuristic to load-balance ad-hoc networks which rely on clusterheads, nodes vested with the responsibility of routing message for all the nodes within their cluster [1]. They distribute the responsibility of being a clusterhead by shifting the load from one node to another. Although over time, load-balancing is achieved, at any given time, there is an unfair burden of memory responsibility placed on these clusterheads. In contrast, GLS2 allows each node to independently decide how much memory to invest in building a location table. It never requires a node to store more location entries than it can fit in its memory.

Chapter 3

GLS2: The New Geographical Location Service

In this section, we present the major design choices in GLS2, the new geographical location service, and the rationale behind these decisions. To distinguish between our new protocol and the original, we will refer to the system which will deploy GLS2 as Grid2.

3.1 Limiting the Table Size

The original GLS operates under the assumption that there is uniform distribution of nodes across the network. It is this assumption that leads the authors of Grid to the claim that the per-node storage cost grows as a small function of the total number of nodes. However, as observed in the scenario presented in Figure 2-2b and Appendix A, a node might be required to serve 80-100% of all nodes in the network to ensure correctness.

We therefore designed a protocol that allows a node to resist unfair memory requirements. Nodes are no longer required to blindly admit all requests for service in their location table to ensure correctness. On the most fundamental level, this requires imposing a limit on the location table size. Intuitively, limiting the table size is a mechanism similar in many ways to cache replacement strategies and admission control; when its table reaches capacity, the location server can selectively drop the new update or replace existing entries.

Limiting the table size thus empowers the location server, enabling it to resist an extreme memory demand in situations of uneven node distribution. Coupled with an appropriate dropping strategy, the location server can establish greater control over which updates it admits to ensure that queries can still be routed efficiently despite dropped location information.

Preferential dropping does not affect the knowledge that a location server has of other nodes in

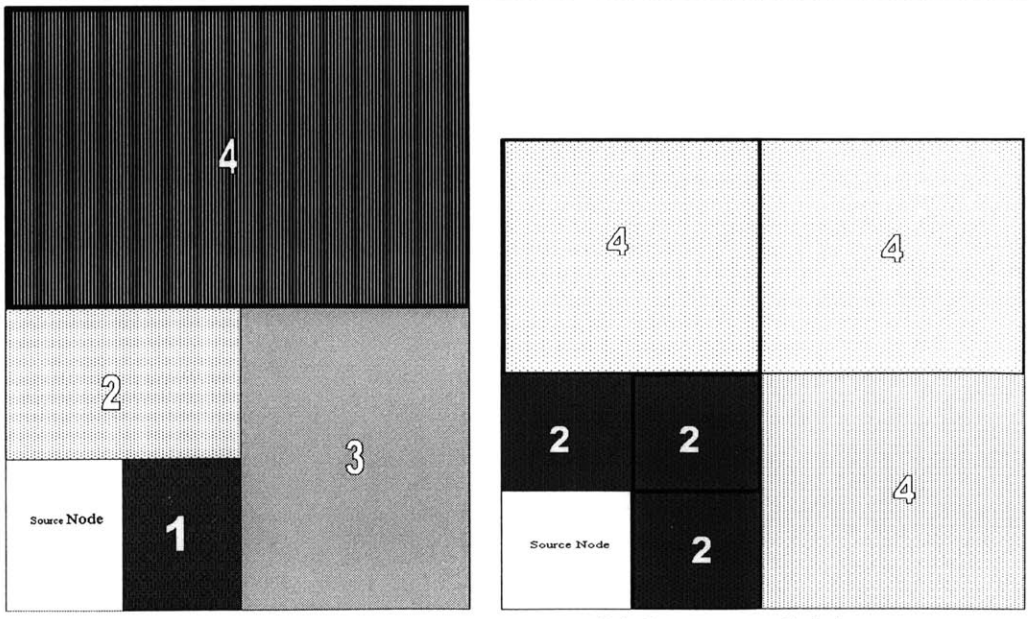
its own smallest grid. This is because a smallest grid is defined to be the size of the coverage area of a single radio-range in which a node can announce itself. Consequently, a node can hear the beacon of each neighbor within its own cell.

3.1.1 Preferential Dropping from the Location Table

We have implemented a dropping strategy that preferentially drops location information of nodes based on geographical location. A location server that has exceeded its storage limit preferentially keeps information about nodes closer to it. This ensures that queries do not travel to a distant location when the destination is actually very close to the source node. Put another way, it ensures that the query will never leave the order-n subdivision for which both the source and destination are members thereby preserving the efficiency of the original GLS. This will be explained in more detail.

3.2 Binary Subdivisions

The authors of the Grid paper admit that quad-dividing the world was an arbitrary choice. In Grid2 we split the world in half at each level, rather than in fourths, by using rectangles with an aspect ratio of $\frac{1}{\sqrt{2}}$. At successive levels, each rectangle may be divided into two such rectangles, these two rectangles being sibling rectangles to one another.



(a) Binary Subdivisions (b) Quaternary Subdivisions

Figure 3-1: The source node's sibling grids at each level.

Our primary reason for choosing to subdivide the network into halves is that it allows a query to converge to either the destination or a location server for the destination in the event of dropped location information in a simpler and more efficient manner than would quadtree partitioning. This will be explained in detail in the next section.

Furthermore, in a network in which memory is a constraint, binary subdivisions is a logical choice. The number of location servers that a node must recruit is equal to the number of sibling grids per-level in the geographic hierarchy multiplied by the number of levels in the hierarchy. For a quaternary based system, this means that a node must maintain $3log_4 n$ servers, in a network that is n times the size of the coverage area of a single radio. However, by subdividing the network in halves, a node need only to recruit on the order of $log_2 n$ servers, where n is the same coverage area of that in quaternary partitioning. To understand this result, consider a radio range of radius r. This will be able to fully sweep over a square of area $2r^2$ or a rectangle of proportions $1{:}\sqrt{2}$ of area $\frac{4\sqrt{2}}{3}r^2$. The number of per-node location servers in the binary system is $lg_2 m$, where m is the number of of rectangles of proportions $1{:}\sqrt{2}$ which can be fully covered by the radio. Thus, in the binary system is each node must maintain $lg\frac{3n}{2\sqrt{2}}$ location servers, where n is the coverage area in the quaternary-based system. For a large n this is approximated with $lg_2 n$.

Let us appreciate the immense benefits gained by choosing a network employing binary subdivisions over quaternary divisions. First, the memory requirement for a binary based system is 33% less. This is because on average each node is required to be a location server for $lg\frac{3n}{2\sqrt{2}}$ nodes as opposed to $3log_4 n$ nodes. Second, the overhead bandwidth required by update packets is also reduced by 33%. This results because each node is only required to maintain $\frac{2}{3}$ of the location servers of its quarternary counterpart.

Unfortunately, there is a necessary tradeoff between memory requirements on the one hand and average query path length on the other. Binary subdivisions have the consequence of a longer average query path length than do quaternary subdivisions in scenarios where no nodes have dropped location information. Grid's location query algorithm using quaternary subdivisions is guaranteed to find the destination in $log_4 n$ query hops whereas our location query algorithm, using binary subdivisions, can take up to $log_2 n$ query hops where source and destination are a distance of n order-0 grids from each other. However, each query request in binary subdivisions has a tighter bound on the distance it must travel at each stage in binary subdivisions than in quaternary subdivisions. This results from the relative location of X's order-n and order-(n+1) grids in the two different networks. In a binary subdivided world, node X's order-n rectangle is located next to its order-(n+1) rectangle, whereas in a quaternary world there is no such constraint. A query traveling from X's order-n square to its order-(n+1) square, might have to travel through one or more of X's other order-n squares to reach its destination. This is apparent in Figure 3-1.

We felt that the 33% reduction in overhead and memory requirements as well as the added

efficiency and simplicity gained for our new protocol significantly outweighed the longer average query path length. In contrast to the query path length overhead which happens only when nodes try to find the location of other nodes, the update overhead in Grid is always present, even if nodes are not seeking location information.

3.3 Our Location Table

Each location server contains a location table which can admit up to M mappings of node IDs to the nodes' current geographic locations. Additionally, each has a *full-flag* a flag indicating the smallest level rectangle for which it has dropped location information. When a server receives a location update, it immediately admits the update if space exists. Otherwise, it drops the appropriate information from its location table as based on the scheme given in section 3.1.1 and updates its full-flag.

3.4 LQA2: Our Location Query Algorithm

When designing this new algorithm, of highest priority was that we preserve the efficiency of GLS. The query algorithm for GLS never leaves the order-n rectangle containing both the source and destination nodes. We also are able to prove that LQA2 will never go outside of its order-n rectangle to find information that resides inside of that rectangle, which we will demonstrate in the next section.

Our new Location Query Algorithm (LQA2) works similarly to Grid's original algorithm for finding a destination as given in section 2.1.4. At each successive step, the forwarding node sends the query to its $CND_{ForwardingNode}$; each node X that receives a location query looks through its location table for a node whose ID is closest to the destination, and closer to the destination than X itself in ID space. However, unlike each intermediate node in GLS, node X is not guaranteed to find a node closer to the destination than X. If X cannot find a CND_X, LQA2 exploits the benefits of binary subdivisions as well as preferential dropping to find the destination. First, X checks its full-flag to see which level it has dropped location information. X knows that it only has full knowledge of the world up to the level for which it has started dropping information and no knowledge of the world outside of this full level. Since there are no nodes closer to the destination than X in any level of which X has full location information, X knows that neither the destination nor location servers for the destination are in any of these levels. Therefore, it forwards the query to X's sibling grid in the unsearched portion of the world knowing that there is nothing in its portion of the world that could get it closer to the destination than itself. The level of the sibling grid is determined by how much information the location server has retained, which grids have already been searched, and proximity to that sibling grid.

Table 3.1: GLS2 Query Packet Fields. Note that the "Searched list" can be expressed with a maximum of $lg_2 n$ entries, where n is the number of smallest grids in the network, as shown in 4.5.)

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
Searched list: Rectangles visited by the query

When looking for the CND_X, only those entries in X's location table that are in grids which the query has not visited are considered for candidacy. This is because either X is the lowest node that the query has reached and hence is the closest node to the destination in the grids which have already been searched, or there exists some other node Z which is closer than X. However, this node must have dropped all location information about any node closer than itself in this level rectangle. This is because if Z would have known about a node closer than itself then it would have sent the packet straight to that node. X (which is larger) would have never been sent the query.

Put in a different light, LQA2 eliminates successive grids from consideration of holding either the destination or location information. It first asks all possible nodes in the subdivisions for which the location server has information if they know any node closer than itself in ID space to the destination. If none does, then it is assumed that the destination's location is known in some other part of the world as determined by our algorithm. This process of elimination proceeds recursively until eventually the destination is found.

The simplicity that binary subdivisions affords is apparent as we consider the necessary steps for a query in a quadtree configuration. Since each node only picks one location server in each quad-partitioning, the query has three alternative "quarters" at each level in which to look for the CND_X, instead of just one. There is no logical progression for where to send the query packet. Since the node does not know which quarter is the right one in which to look for destination or location servers for the destination, it must chose one at random. In contrast, a node in a binary subdivided world has but one logical choice: it forwards the packet to its only sibling grid at the appropriate level of the grid hierarchy.

Chapter 4

Correctness and Efficiency

We have designed our new location query algorithm to be both correct and efficient. In this section, we will explain what we mean by efficiency, give an intuition for LQA2, state the invariants of our algorithm, and give a formal proof of the correctness of LQA2.

Although our proof is for queries sent in static static networks, simulation results show that the algorithm is extremely effective in dynamic environments as well.

4.1 Efficiency

The LQA2 algorithm is efficient in two main regards. First, a location query from **A** to **B** will never go outside of the smallest rectangle subdivision that contains both **A** and **B**; a query will first ask a location server physically closer to the source node for the destination before being forwarded to far away location servers. Practically, this means that a query must search **A**'s order-n rectangle for any node with ID closer to the destination which could possibly route the query to a node with ID closer to the destination before searching its order-(n+1) rectangle.

Second, each step of the query will search only rectangles which have not yet been searched. This ensures that at each successive query step, the query will search within rectangles which have the possibility of holding destination information.

Unless it arrives at the destination earlier, the query will not terminate until it has searched every rectangle in the universe.

The above three claims lead to the conclusion that the query will terminate at the destination without traveling outside of the order-n rectangle for which both source and destination are colocated.

4.2 Intuition

Let us build an intuition for LQA2 by understanding the steps a query takes to find the destination if all location servers have dropped information about their charges except neighbors colocated in their order-0 rectangle. Without loss of generality, we assume that the destination has an id of 0. A query in which nodes have infinite memory capabilities, as in the original GLS, will always visit the smallest node in each level. See [8]. (This is because the smallest node in each level will either be the location server for 0, the location server for some node smaller than itself, or the destination itself as per the original Grid algorithm.) Our new algorithm differs in that we must now deal with possible dropped location information. LQA2 also search for the smallest node in each level, but with one caveat. LQA2 only search for the smallest node which could contain information enabling the query to travel closer to the destination than it already is. Therefore, the query is always looking for the smallest ID node out of all the rectangles searched which could possibly hold information about some node smaller than itself in an unsearched portion of the network.

The new protocol is as follows: The node that receives a query will search for a location server smaller than itself in a rectangle which it has not yet searched. If it finds one, it will send the query to that node. Otherwise, the query is sent to the *current source*'s lowest order sibling rectangle which has not been searched. The query is then restarted from this new rectangle from a random node in that rectangle. This node becomes the new *current source*.

In Figure 4-1, the query starts in the lowest left corner rectangle at the original source. The original source is also the first *current source*. The original source, CS1, knows of no node smaller than itself. Since CS1 does not have information about its order-0 sibling rectangle, CS1 sends the query to its order-0 sibling rectangle. The query restarts from the first node in this rectangle to receive the packet. Lets call this node CS2. CS2 is the new *current source* for the query. CS2 knows about some node in its order-0 rectangle that is smaller than itself, the *SNIR* (smallest node in this rectangle), so CS2 forwards the query to this *SNIR*. However, this *SNIR* knows about no node smaller than itself since it has dropped all information above its order-0 rectangle. Therefore, this *SNIR* sends the query to CS2's lowest order sibling rectangle which has not been searched. CS2's order-0 sibling rectangle has already been searched, therefore the query is forwarded to CS2's order-1 sibling rectangle. The query restarts from the first node in this rectangle to receive the packet. Lets call this node CS3. The query continues in this manner until it arrives at node 0, the final destination.

4.3 Invariants

At each stage of the query, GLS2 ensures that these three invariants hold true:

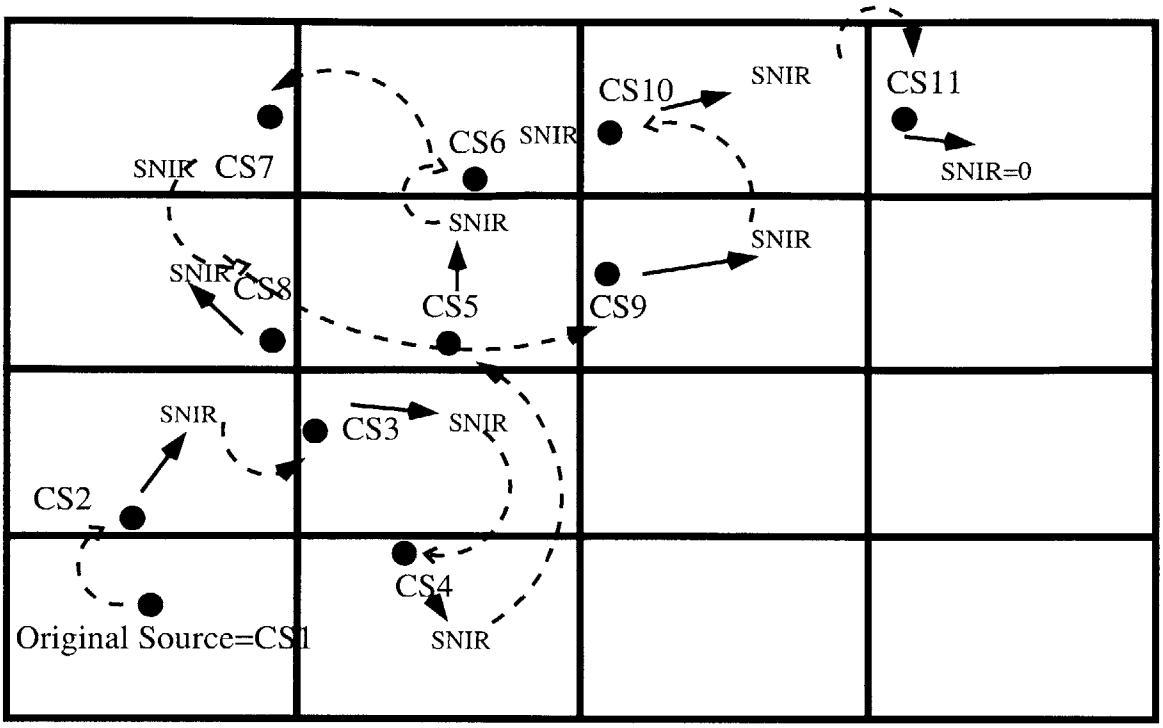


Figure 4-1: Route query takes to find the destination in a network in which location servers have dropped all of their charges except its neighbors in its order-0 rectangle. Each *current source* is labeled as *CS*. The *SNIR* is the node with the smallest ID in each specific rectangle. Straight lines indicate a direct forwarding to a smaller node. Curves indicate forwarding the query to the current CS's lowest order sibling rectangle which has not been searched to restart the query.

- 1: The query will search rectangles which have not yet been searched.

- 2: The query searches the *current source*'s sibling rectangles in monotonically increasing order. The *current source* is defined to be either the original source (if the query has never been restarted from a new node) or the most current node at which the query has been restarted.

  Note that because the current source's sibling rectangles are searched in monotonically increasing order, the query will always search the original source's order-n sibling rectangle before searching its order-(n+1) sibling rectangle. Therefore, the query will never travel outside of the order-n rectangle for which both source and destination are colocated.

- 3: The query will reach the *CND-CPHI* of the rectangles searched. The *CND-CPHI* is the *Closest Node to the Destination which Could Possibly Hold Information* about some node closer than itself in ID to the destination in an unsearched portion of the network.

4.4 Proof of Correctness

We will prove that, in a static network, GLS2 routes every location query to its correct destination by proving that our three invariants always hold true in a network in which nodes have full knowledge of all neighbors in their order-0 rectangle. Our proof works by induction.

We assume, without loss of generality, that our destination node has an ID of 0. In this case, the *CND-CPHI* is the smallest ID node in all the rectangles searched which could possibly hold information about some node smaller than itself in an unsearched portion of the network.

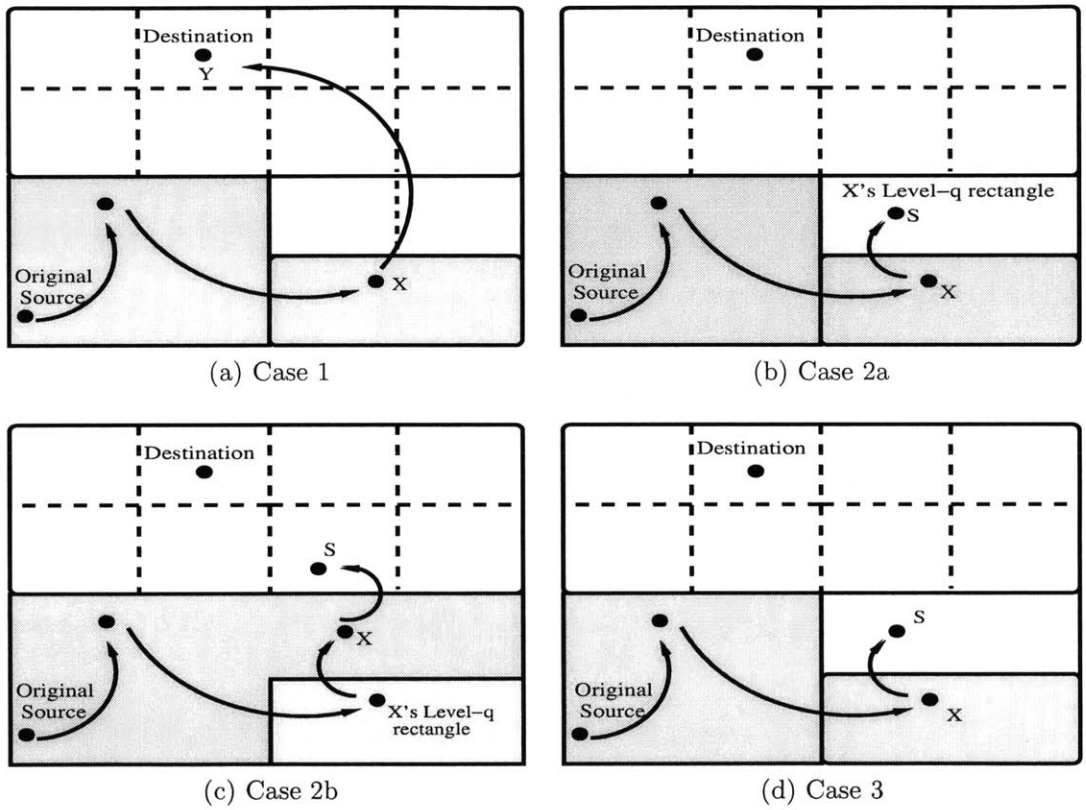


Figure 4-2: Routing query packet depending on case: The query has checked dark gray areas for destination information.

Base case (First Query Step)

Suppose the query begins at a node X, node X may or may not be the node with the smallest ID in its order-0 rectangle. If so, the query trivially reaches the smallest node in the order-0 rectangle after zero location query steps. In particular, if X is not the node with the smallest ID, then X will know the location of the node with the smallest ID in the order-0 rectangle, Y, because of our base case assumption that X should know every node in its order-0 rectangle. Therefore, the query will

be forwarded to the smallest node in the order-0 rectangle of the source.

It is trivial to see that all three of our invariants hold.

- 1: Query began here so obviously never searched the rectangle at this level.
- 2: The query will search the order-0 rectangle that contains the source before all others.
- 3: The query will terminate at the *CND-CPHI* which is the smallest node in this order-0 rectangle.

Inductive step (Kth query step): The query has finished searching the current source's order-n sibling rectangle and has not searched the current source's order-(n+1) sibling rectangle. We call the current source 'CS'. The query is at node X.

Case 1: X knows about some node Y whose ID is smaller than itself in CS's order-z sibling rectangle. The query has not yet searched this rectangle ($z > n$).

X knows the location of Y and will not know the geographical position of any node whose id is smaller than Y outside CS's order-z sibling rectangle. 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 order-z sibling rectangle because X's order-z sibling rectangle is Y's order-z sibling rectangle. Node Y must have selected X because X is the smallest node in its order-z sibling rectangle and hence is Y's successor in that rectangle. Node X will not know the location of any node lower than Y outside of its order-z sibling rectangle because when any such node sought a location server in X's order-z sibling rectangle, Node Y was the better choice. Therefore the smallest node that X is aware of is Y and the query will be forwarded there. The query finished searching all rectangles in CS's order-z sibling rectangle and will proceed to search CS's order-(z+1) sibling rectangle, which has never been searched before.

Our three invariants follow directly from the correctness of Grid in [8]:

- 1: We know from our assumption that CS's order-z rectangle has never been searched.
- 2: The query searched CS's levels in monotonically increasing order. It has searched all rectangles in CS's order-(z+1) rectangle, but nothing beyond.
- 3: The query is now at the *CND-CPHI* in CS's order-(z+1) rectangle.

Case 2:

- **X knows of no node smaller than X that has not already been searched and**
- **X has dropped location information about nodes in its order-q sibling rectangle** ($1 \leq q \leq$ MAX LEVELS).

Case 2a :The query has never searched X's order-q sibling rectangle.

In this case, X routes the packet to any node in X's order-q sibling rectangle and restarts the query from this node. Call this node S. X knows that there are no nodes smaller than itself which have not already been searched in its order-(1...q-1) sibling rectangles. Any node Y smaller than X in X's order-k sibling rectangles for all k, $1 \leq k \leq q-1$, must have selected a location server in X's order-k sibling rectangle because X's order-k sibling rectangle is Y's order-k sibling rectangle. Node Y must have selected either X or some other node which dropped location server information. This is because either X is the smallest node in its order-k sibling rectangle and hence is Y's successor in that rectangle, or there exists some other node Z which is smaller than X in X's order-k sibling rectangle. However, Z must have dropped all location information about any node smaller than itself in CS's order-(n+1) sibling rectangle. If Z would have known about a node smaller than itself in CS's order-(n+1) sibling rectangle, then Z would have sent the packet straight to that node. X (which is larger than Z) would have never been sent the query.

We deduce from the proof that our three invariants hold true:

- 1: X's order-q sibling rectangle has never been searched as a given.
- 2: The query's current source is either the same one which routed the query to X and S, or it is X itself. Therefore, $(q \geq n+1)$ because CS's entire order-n rectangle has already been searched. Thus, the query has now searched all rectangles up to CS's order-q rectangle, but nothing beyond.
- 3: Since X was the previous *CND-CPHI*, but X did not know of any node smaller than itself in a rectangle which had not been searched, S is now the *CND-CPHI* regardless of its node ID.

Case 2b: The query has searched X's order-q sibling rectangle.

X routes the packet to a node in CS's lowest order sibling rectangle which has not been searched. It restarts the query from any node in this rectangle. Call this node S. X knows that there are no nodes smaller than itself which have not already been searched in CS's order-k sibling rectangles, for all k, $1 \leq k \leq q-1$. Any node Y smaller than X in X's order-k sibling rectangle must have selected a location server in X's order-k rectangle because X's order-k rectangle is Y's order-k sibling rectangle. Node Y must have selected either X or some other node which dropped location server information. This is because either X is the smallest node in its order-k sibling rectangle and hence is Y's successor in that rectangle or else there exists some other node Z which is smaller than X in X's order-k sibling rectangle. However, Z must have dropped all location information about any node smaller than itself in CS's order-k rectangle. If Z would have known about a node smaller than itself in CS's order-(n+1) sibling rectangle, then Z would have sent the packet straight to that node. X (which is larger than Z) would have never been sent the query.

- 1: As per our assumption the query is searching CS's lowest order rectangle which has not been searched. (Unless there was a previous source which searched CS's order-(n+1) rectangle, the query will be routed to this level.)

- 2: The query is searching CS's levels in monotonically increasing order. (It is searching the lowest order rectangle which has not been searched before searching its higher level rectangle.)

- 3: Since X was the previous *CND-CPHI*, but X did not know of any node smaller than itself in a rectangle which had not been searched, S is now the *CND-CPHI* regardless of its node ID.

Case 3: X knows of no node smaller than X that has not already been searched and X has not dropped any information.

X routes the packet to a node in CS's lowest order sibling rectangle which has not been searched. It restarts the query from any node in this rectangle. Call this node S. X knows that there are no nodes smaller than itself which have not already been searched in CS's order-k sibling rectangles, for all k, $1 \leq k \leq n$. Any node Y smaller than X in X's order-k sibling rectangle must have selected a location server in X's order-k rectangle because X's order-k rectangle is Y's order-k sibling rectangle. Node Y must have selected either X or some other node which dropped location server information. This is because either X is the smallest node in its order-k sibling rectangle and hence is Y's successor in that rectangle or else there exists some other node Z which is smaller than X in X's order-k sibling rectangle. However, Z must have dropped all location information about any node smaller than itself in CS's order-k rectangle. If Z would have known about a node smaller than itself in CS's order-(n+1) sibling rectangle, then Z would have sent the packet straight to that node. X (which is larger than Z) would have never been sent the query.

- 1: The query has been routed to a rectangle which has not yet been searched. (Unless there was a previous source which searched CS's order-(n+1) sibling rectangle, the query will be routed to this level.)

- 2: The query is searching CS's levels in monotonically increasing order. (It is searching the lowest order rectangle which has not been searched before searching its higher level rectangle.)

- 3: Since X was the previous *CND-CPHI*, but X did not know of any node smaller than itself in a rectangle which had not been searched, S is now the *CND-CPHI* regardless of its node ID.

Because our query searches a new rectangle at each query step in monotonically increasing order of levels, the query will reach the destination.

4.5 Increased Packet Size

LQA2 has an extra expense in that it requires the query packet to record the rectangles for which it has searched to ensure that the query does not revisit a rectangle. However, we maintain that this requirement does not add many more bits to the original query packet. Indeed, the query packet must keep track of at most $lg(n)$ entries, where n is the total number of order-0 rectangles. This is because when the query leaves the source's order-n rectangle, it needs only one entry containing that level rectangle. This is a result of the invariant that the query packet searches the source's entire order-n rectangle before searching a rectangle outside of it. If, however, a query has searched the source's order-n rectangle but not yet finished searching its order-(n+1) rectangle, it must store the rectangles visited in this *intermediate zone*. We know, however, that there are only a total of n possible levels inside this *intermediate zone*. As each of these levels requires exactly one entry, a maximum of n, where $n \leq n$ entries must be appended to a query packet. Since the maximum number of levels is $lg(n)$, the maximum number of entries appended to a query packet is $lg(n)$.

Chapter 5

Implementation

We begin this chapter with a short overview of our code modifications. We then describe our simulation environment and conclude with a discussion of the various scenarios with which we conducted experiments.

5.1 Modifications

To implement our proposed protocol we modified the existing GLS simulation code which is comprised of C++ and TCL code that runs over the *ns* Network Simulator. Our modifications to the original code, as obtained from [9], consisted of changing all instances of quaternary subdivisions to binary subdivisions, implementing a preferential dropping of location information from location tables, as well as deploying our new location query algorithm. We preserved all other procedures from the original Grid to ensure accurate protocol comparison.

5.1.1 Location Updates

The nodes in both Grid and Grid2 update their order-i servers by sending out update packets after each movement of $2^{i-2}d$, where d was set to 100 meters. To properly compare Grid2 and Grid we updated the same area of each at equitable rates. More specifically, the nodes in Grid2 updated their odd-level servers at a rate equal to the next highest level. To understand this, note that in Figure 3-1, the area covered by Grid's level-2 sister squares is equal to the area covered by the combination of Grid2's level-1 sibling grid and level-2 sibling grid and are therefore updated at the same rate.

As explained in the Grid paper, this scheme ensured that a node sent updates at a rate proportional to its speed and distance to its servers. In addition to sending its location information, a node alerted each of its location servers to the expected time that it would again update this server. The

server utilized this information by invalidating all location information which had timed-out. This ensured that a location server gave only fresh information.

5.1.2 Geographical Forwarding

We used a similar geographic forwarding layer to Grid. The geographical forwarding layer employed a two hop distance vector protocol. For more detailed information see [8].

5.1.3 Preferential Dropping

To accurately compare Grid to Grid2, we implemented location based preferential dropping in both protocols. If a location server reaches its storage capacity, it preferentially dropped location information about nodes in its higher order subdivision, i.e. it dropped its more distant charges. Choosing between information of nodes that are located in the same rectangle, a server dropped the entry with the smallest timeout value. This ensured that the information kept by a server was the most up to date.

When space was available, a location server kept any location information which it had recently seen. This might include the location of a node which has initiated contact or location information observed when acting as a forwarding agent. These location entries are called cached entries and are useful in increasing robustness. In the original GLS, a node had an infinite amount of cache storage space. In our new protocol, memory capacity represents the total amount of location entries in both the location table and the cache. Given the choice between keeping location information for nodes which it must serve and keeping auxiliary information in its cache, a location server dropped the cache entry.

5.2 Simulation

The nodes used the IEEE 802.11 radio and MAC model provided by the CMU extensions [4]. As per the original Grid, the radio's range is set to a radius of approximately 250 meters. We experimented with the effects of changing the maximum location table size per node on different node layouts and travel patterns.

5.3 Scenarios

As per the advice of the original GLS paper, each of our simulations ran with an average density of approximately 100 nodes per square kilometer. We compared the performance of both protocols on a variety of scenario configurations as is explained in this section. All configurations ran for 300

seconds. Over the course of the run, each node initiated an average of 15 location queries to random destinations.

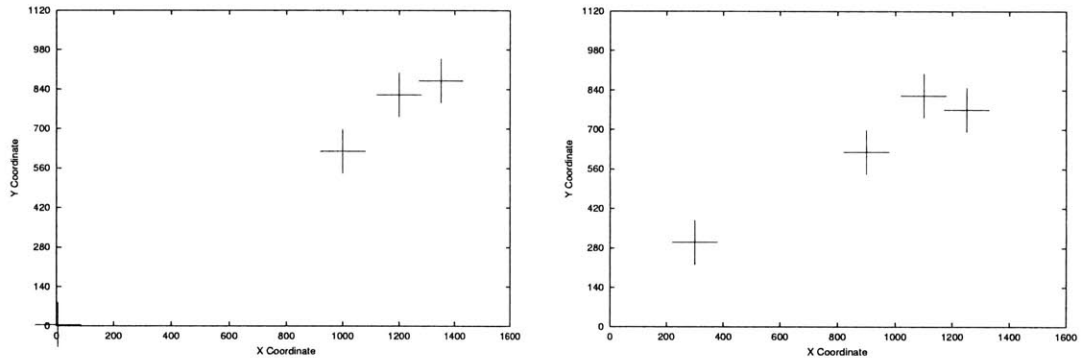
To gain a thorough insight to the power of GLS2, we ran our simulations on networks of both 200 and 500 nodes, each with a different density of nodes. For a network of 200 nodes, the grid hierarchy contained 64 order-0 grids in a universe of 1340 square meters. For a network of 500 nodes, there were 256 order-0 grids in a universe of 2680 square meters. The original Grid system ran on a universe of square formation, whereas our new protocol ran on networks with sides of proportion $1{:}\sqrt{2}$.

Our first configuration consisted of nodes which were uniformly distributed in the network and which movements were determined by the random waypoint mobility model [2]. Each node was placed at a random location at the start of the simulation and moved toward a random destination. A node moved toward its destination with constant speed chosen uniformly between zero and 10m/s. Upon arrival, the node chose another random destination as well as a new speed and moved toward its new goal. These runs did not include pause times.

The other scenarios consisted of a combination of densely populated regions and sparse areas. Each node moved according to a random waypoint model. However, instead of choosing a starting position and destination at random, a node selected these locations from within certain pre-defined regions. Within these respective regions, nodes are distributed uniformly. This is intended to simulate mobile users who move between points of atractions.

We tested two different layouts of nodes distributed in this manner. The approximate location of the regions were chosen to ensure that geographical forwarding would continue to work as it requires a high spacial density of nodes. In both scenarios $\frac{3}{4}$ of the nodes were in the upper right quadrant of the grid while the remaining formed a group in the lower left fourth of the largest rectangle. We accomplished this by making four smaller rectangular regions each of size 47 square meters. Three regions were placed in arbitrary locations in the upper left fourth of the network hand corner and the last arbitrarily in the lower right corner. Given are the locations for the densely populated areas in Grid2. Grid has corresponding densely populated locations, properly calibrated for a square network.

In the first scenario, which we will call *Non-Uniform-1*, the lower left corner of the regions were located at (5,5), (1000,620), (1200,820), and (1350,870). In *Non-Uniform-2*, the lower left corner of the regions were at (300,300), (900,620), (1250,770), and (1100,820).



(a) Non-Uniform-1 (b) Non-Uniform-2

Figure 5-1: Layout of each spacially non-uniform node network. A point denotes a densely populated region. Nodes traveled between regions.

Chapter 6

Simulation Results

In this chapter we present an analysis of the data we gathered in our simulations. Each of the results in our graph come from an average of 5 runs. Generally, the results of our simulations support the theoretical analysis of GLS2 presented in this paper.

6.1 Performance Comparison

We first present results towards evaluating how GLS2 improves on GLS in the most fundamental case: uniform node distribution. We then analyze the results for non-uniformly distributed networks.

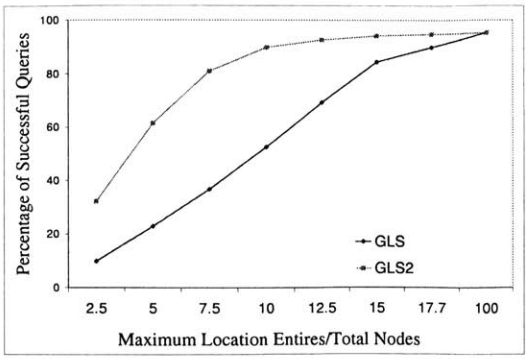
6.2 Uniformly Distributed Node Networks

6.2.1 Query Success Rate

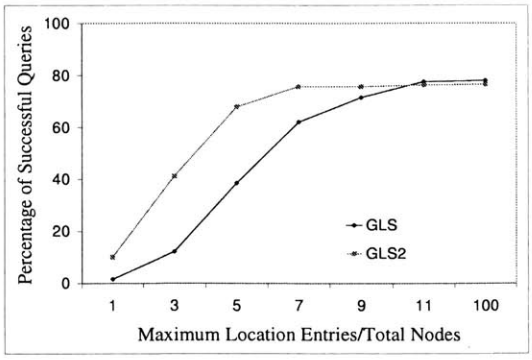
Shown in Figure 6-1 is the percentage of queries successfully routed to their destination as a function of the maximum location table size over the total number of nodes. With infinite memory capabilities, both protocols performed equally. As the number of location entries per server decreased, GLS began to drop packets in an almost linear fashion whereas GLS2 continued to route a significant number of query packets to their the proper destinations.

6.2.2 Distribution of Failures

Not only does GLS2 have a much larger query success rate than GLS in situations of finite location table size, but the distribution of query error types in GLS2 mirror those of infinite memory capacity. Tables 6.1 gives the distributions for the various error types of query failures. Note the very low percentage of failures that the queries of GLS2 encounter because of knowing of no node closer to the destination than itself.



(a) Uniform Distribution of 200 Nodes



(b) Uniform Distribution of 500 Nodes

Figure 6-1: Percentage of queries successfully routed to their destinations in scenarios of uniformly distributed nodes as a function of the maximum number of entries allowed in location table over the total number of nodes.

This distribution gives us a deeper insight into the power of GLS2. Many of the query failures that GLS2 encounters could be alleviated by an improved geographical forwarding strategy. This is due to the fact that a great proportion of "no route", "loop", and "ttl expiration" errors might be solved by a reasonable recovery strategy such as that given in [3] while failures encountered by GLS in situations of limited memory will remain.

Table 6.1: Distribution of Location Query Failures for the Given Maximum Percentage of Nodes Allowed in the Location Table.

Reason for Failure	*GLS*		*GLS2*	
Maximum Percentage of Nodes Allowed	*7.5% (15 Entries)*	*100%*	*7.5 (15% Entries)*	*100%*
Loop, No Route, TTL Expired	2.2%	63.5%	66%	79.8%
No Closer Node	98.8%	36.5%	33%	20.2%

(a)Networks of 200 Uniformly Distributed Nodes

Reason for Failure	*GLS*		*GLS2*	
Maximum Percentage of Nodes Allowed	*3% (15 Entries)*	*100%*	*3% (15 Entries)*	*100%*
Loop, No Route, TTL Expired	5.0%	53.6%	90.2%	73.8%
No Closer Node	95.0%	46.4%	9.8%	26.2%

(b) Networks of 500 Uniformly Distributed Nodes

6.2.3 Bandwidth Consumed By Location Updates

Confirming the claim made in section 3.2, the bandwidth required to update location servers in Grid2 was only $\frac{2}{3}$ that of the original Grid. Table 6.2 presents the average number of update packets sent into the network to maintain a properly functioning location service.

Table 6.2: Average Number of Update Packets Sent in Scenarios of Uniformly Distributed Nodes and Infinite Memory Capacity.

	200 Nodes		*500 Nodes*	
Location Service	*GLS*	*GLS2*	*GLS*	*GLS2*
Average Update Packets Sent	23103	15021	72022	45863
Standard Deviation	910	443	1212	1152

6.2.4 Query Hops

As described in section 3.2, there was an increase in average query hop length when using binary subdivisions as opposed to quaternary subdivisions. Indeed, in the quaternary simulations, on average, each query in a network of 200 nodes had only 69% the query hop distance to travel than each query in a binary subdivided world. However, when analyzing the average total number of geographical forwarding hops per query (i.e., average number of nodes traversed by a query), we find that the discrepancy between the two protocols diminished. This is very significant as each hop requires network resources. To summarize the results, each query in the quad-divided world had 84% the hop distance to travel than a query in Grid2. In the networks of 500 nodes, we found a similar, but not as substantial reduction. On average, each qeuery in the quarternary simulations had 71% the query hop distance to travel but 75% of the total number of query hops. These results can be viewed in Table 6.3.

Table 6.3: Average Number of Query Hops per Query for Scenarios of Uniformly Distributed Nodes and Infinite Memory Capacity.

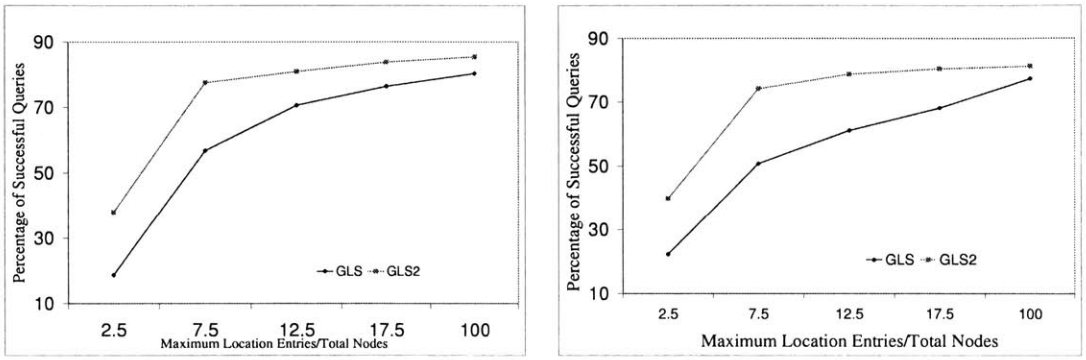
	200 Nodes		*500 Nodes*	
Location Service	*GLS*	*GLS2*	*GLS*	*GLS2*
Number of Query Hops	3.3	4.8	6.5	9.2
Standard Deviation	0.2	0.5	0.4	0.7
Total Number of Query Hops including Geographical Forwarding	6.2	7.4	9.5	12.6
Standard deviation	0.3	0.5	1.6	1.6

6.3 Non-Uniform Distribution of Nodes

6.3.1 Query Success Rate

In addition to studying the query success rates in networks of uniform node distribution, we analyzed data gathered from simulations of non-uniformly populated networks. Note that because nodes are not evenly distributed over the network, there were far greater update packet losses due to holes in

the geographical forwarding layer. As a result, many location servers did not receive these updates from their charges, thus the higher degree of failure in situations were nodes had unlimited storage. Observe the 5% increase in the success rate of queries routed by GLS2 over those routed by GLS. Here we see the additional robustness gained by the GLS2 protocol. In situtations of location update loss due to congestion or network holes, GLS2 can still route the query packet to its destination. This is the result of GLS2's location query algorithm, which is designed to send the query packet to a neighboring rectangle in the situation that a server cannot find a node closer to the destination than itself.



(a) Non-Uniform-1 (b) Non-Uniform-2

Figure 6-2: Percentage of queries successfully routed to their destinations in scenarios of unevenly distributed nodes as a function of the maximum number of entries allowed in location table over the total number of nodes.

Chapter 7

Future Work and Conclusions

7.1 Future Work

There are many areas of future work opportunities that further enhance Grid2's capabilities. Firstly, developing an improved greedy forwarding strategy might considerably improve GLS2's performance. As mentioned in section 6.2.2, most of the query failures resulted from errors in the geographical forwarding layer. The design of an enhanced recovery strategy could mitigate the amount of packets dropped because of loops or dead ends.

Additionally, similarly to the original Grid, in our current implementation, a query from one node to its geographically close neighbor might have to traverse the entire network. This is the result of the grid hierarchal structure; nodes located on two different sides of the largest grid boundary, even if they are geographically very close to one another, are considered to be far. It would be advantageous to ensure that a query sent from one node to its geographically close neighbor never travels far distances.

7.2 Conclusions

In this paper, we presented a new geographical ad hoc routing protocol that improves on the Grid system's scalability while still preserving its fundamental design. In uneven node distributions, Grid places undue stress on the nodes located in a more sparse region of the network which adversely affects its scalability capabilities. Grid2 is designed to route packets to their proper destinations while allowing nodes to resist extreme memory requirements.

Initial simulation results comparing the two protocols demonstrate that our new location query algorithm coupled with binary subdivisions is a significant improvement on the original protocol. Not only does Grid2 require $\frac{1}{3}$ less bandwidth to maintain its servers, its performance far surpasses

Grid's in situations where each node is allocated only a finite amount of memory.

We feel that Grid2 holds great promise as a practical and scalable ad-hoc routing protocol.

Appendix A

Mapping of Location Servers to their Charges

Table A.1: Mapping of Location Servers to their Charges.

Node	*Nodes in Location Table*	*% of Nodes Serving*
14	1,2,3,4,5,6,7,8,9,10,11,12,13,78,81,87,95,97,99,101,113	35%
16	1,2,3,4,5,6,7,8,9,10,11,12, 13,14,15,23,24,25,31,32, 33,34,35,36,41, 43,44,45,47,49,51,52,56,57,59,63,66, 66,68,70,71,73, 74,77,78,81,87,95,97,99,101,113	85%
22	17,18,19,20,21	8%
78	14,16,15,22,23,24,25,31,32,33,34,35,36,41,43,44,45,47, 49,51,52,56,57,59,63,66,68,70,71,73,74,77	53%
97	All nodes (except itself)	98%

In Table A each sparse node is mapped to the nodes for which it is a location server and the percentage of total nodes that it serves. Four out of five sparse nodes serve a significant percentage of nodes. This would be unacceptable in a large network.

Bibliography

[1] Alan D. Amis and Ravi Prakash. Load-Balancing Clusters in Wireless Ad Hoc Networks. *Proc. of 3rd IEEE Symposium on Appliaction-Specific Systems and Software Engineering Technology*, pages 25–32, March 2000.

[2] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. *Proc. MOBICOM'98*, pages 85-97, 1998.

[3] Douglas S. J. De Couto and Robert Morris. Location Proxies and Intermediate Node Forwarding for Practical Geographic Forwarding. *MIT Laboratory for Computer Science Technical Report MIT-LCS-TR-824*, June 2001.

[4] CMU Monarch Group. CMU Monarch extensions to ns. `http://www.moarch.cs.cmu.edu`, August 2003.

[5] Hannes Hartenstein, Michael Kasemann, Holger FuBler, and Martin Mauve. A Simulation Study of a Location Service for Position-Based Routing in Mobile Ad Hoc Networks. *REIHE INFORMATIK*, July 2002.

[6] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. *Proc. 29th ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[7] Michael Ksemann and Hannes Hartenstein. Analysis of a location service for position-based routing in mobile ad hoc networks. *WMAN 2002*, pages 121–133.

[8] Jinyang Li, John Jannotti, Douglas S. J. DeCouto, David R. Karger, and Robert Morris. A Scalable Location Service for Geographic Ad Hoc Routing. *ACM Mobicom 2000*, pages Boston, MA.

[9] The Grid Ad Hoc Networking Project page. URL: `http://pdos.lcs.mit.edu/grid`, August 2003.

[10] USCG Navigation Center GPS page. `http://www.navcen.uscg.gov/gps/default.htm`, August 2003.

[11] R. Prakash, Z. Haas, and M. Singhal. Load-balanced location management for mobile systems using quorums and dynamic hashing. *Baltzer Wireless Networks (WINET) Journal*, 7(5):497–512, September 2001.

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