EFFICIENT CONTENT LOCATION IN MOBILE AD HOC NETWORKS

BY

JIVODAR BORISLAVOV TCHAKAROV

B.S., University of Louisiana at Lafayette, 2000

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2003

Urbana, Illinois

Abstract

The rise of computer networks has led to a new paradigm of computing. The standalone, isolated computer has given way to the computer as part of an infrastructure working together to provide users with increased benefits. This has led to an abundance of information and services made available to users by remote servers. A fundamental issue in this environment is efficiently locating needed content. Such content may be in the form of files, services, or any other kind of data. A number of strategies have been proposed to tackle the problem; however, most of these strategies assume a resource-rich network. This assumption does not hold in the case of mobile ad hoc networks which have different characteristics and requirements.

In this thesis, we describe an algorithm for efficient content location in location-aware ad hoc networks. The Geography-based Content Location Protocol (*GCLP*) makes use of location information to lower proactive traffic while minimizing query cost. This protocol increases the amount of proactive location advertisements while limiting the cost of location queries. The protocol is based on geographical information coupled with simple geometric rules. The results of our analysis show that *GCLP* performs favorably in terms of overhead, latency, and scalability.

To my parents and grand-parents.

Acknowledgments

This thesis would not have been possible without the encouragement and help of my parents, Boris and Tania Tchakarov, and grand-parents, Maria and Jivko Tchakarovi and Mariana and Todor Staykovi. They instilled in me the desire for education and pushed me every step of the way. Sadly, not all of them could be here to share my happiness at this success. Thanks to William List for providing an idea which is essential to the development of the protocol described in this thesis and to Dr. Nitin Vaidya for the help and guidance.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

GCLP Geography-based Content Location Protocol.

CS Content Server.

CLS Content Location Server.

HM Hello Message.

UM Update Message.

QM Query Message.

RM Response Message.

MANET Mobile Ad hoc Network.

P2P Peer-to-peer.

SLP Service Location Protocol.

CAN Content-Addressable Network.

GSD Group-based Service Discovery protocol.

AODV Ad hoc On-demand Distance Vector.

RP Resource Provider.

DA Discovery Agent.

Chapter 1

Introduction

Since the emergence of the Internet, a new level of connectivity has made itself present in everybody's daily life. It has made sharing of data easier than ever with a few simple clicks of the mouse button. The rise and fall of a popular file sharing service such as *Napster* [1] has led to the need for new and more creative protocols for location and sharing of data. While previous approaches were following the client-server model, the need for avoiding centralized responsibility and increasing stability has been the driving force behind a variety of new approaches that have been set forth in the context of peer-to-peer file sharing.

The rise of mobile computing has further increased the pervasiveness of devices capable of storing data and requiring the ability to efficiently locate content available on the Web. Cell phones and wirelessly connected PDA's have become a new kind of storefront for e-tailers. Such technical novelties have further increased the need to efficiently locate not only general content but specific services available on the net-work. Printers, for example, are a common need for a variety of applications. Given a pervasive wireless network, the user may not always be expected to know the location and characteristics of the device closest to him. To deal with this problem, a variety of algorithms have been presented to solve the problem of resource location in ad hoc networks. This paper explores current content location approaches and proposes an adaptable protocol for efficiently providing content location in a infrastructure-less network.

File-sharing services have made content location a reality in today's networks. Centralized directory-based content location protocols [1] were the dominant schemes due to their higher reliability and increased possibility of control. However, ownership rights and litigation have prompted a move away from centralized approaches towards decentralized, peer-to-peer architectures. A

number of recent proposals [2, 3, 4, 5, 6] exemplify this trend.

In the last decade, a new type of network has gained momentum. The mobile ad hoc network (*MANET*) is not dependent on an underlying architecture as its predecessors. The mobile nodes making up the network are dynamically and arbitrarily located in such a way that connections between nodes may change on a continual basis. Unlike traditional networks, the ad hoc network does not rely on preexisting infrastructure. Instead, all nodes in the network collaborate to transfer data between points in the network [7]. Such networks are a necessity in environments such as a natural disaster area or a military operation, where no assumptions can be made about any preexisting infrastructure. While, ad hoc networks provide additional flexibility when compared with traditional networks, they also have increased cost of operation that must be considered in designing any protocols that operate on top of such topologies. In such an environment, resources such as bandwidth and power are extremely limited. Of primary importance is the need to keep the amount of resources used during any operation to a minimum.

A final factor is the growth of the Global Positioning System (GPS) [8]. This system has enabled mobile devices to know their exact geographic location and has spawned a number of protocols that make use of geographic information to provide efficient routing in ad hoc networks [9, 10]. More recently, algorithms have been proposed that make use of such location information to provide content discovery to users in the network [11, 12].

A variety of content location protocols have been proposed in recent years. These can be categorized into two major groups - centralized and peer-to-peer systems. Centralized systems [13, 14] depend on a central directory server which will handle the content location on behalf of a requesting client. Such approaches are easier to implement and generally more reliable than the strictly peer-to-peer protocols as the central server is considered to have universal knowledge of the content available on the network. Peer-to-peer (*P2P*) approaches [2, 3, 4] do not depend on a central server. Instead, the nodes in the network collaborate to provide the desired content location service.

A common assumption made by the majority of both centralized and P2P protocols is the abundance of resources in the network, most notably bandwidth. This assumption fails in an ad hoc network and an entirely different approach is necessary. Several protocols have been proposed

to solve the problem of content location in ad hoc networks [15, 16, 17]. However these solutions do not take sufficient care to lower the protocol overhead as they either depend on broadcasting information throughout the network or do not take into account link costs.

In this thesis, we present a content location service, the *Geography-based Content Location Protocol* (GCLP) that takes geographical information into account to provide an efficient content location service to nodes in an ad hoc network. GCLP assumes that all devices in the network know their own location. Since GCLP is meant to operate in an ad hoc network, it is important to summarize some of the properties of the environment as they relate to the content location service. First, we cannot assume a static topology. Nodes may join and leave the network at any time and node mobility is an accepted occurrence. Second, the cost of making sure that everyone knows about everything is prohibitive. Thus, to locate a specific content, a device need not be aware of all content available on the network. In such an environment, GCLP nodes make use of geographic information to periodically advertise content they are hosting to nodes along several geographical directions. Nodes that attempt to locate content need only contact one of these nodes to become aware of the presence of the desired content on the network and the closest available server.

The rest of the thesis is organized as follows. In Section 2 we discuss related work in the area of content location. Section 3 contains a list of requirements we impose upon our final design for GCLP. Section 4 describes the details of our GCLP protocol which is then analyzed in Section 5. Simulation results of GCLP are provided and examined in Section 6. Section 7 proposes several possible optimizations to the protocol. We conclude the work of this thesis in Section 8.

Chapter 2

Related Work

The abundance of information in today's networks has led to the need for efficient and fast mechanisms for discovery of various types of content. A number of approaches have been proposed that attempt to solve this problem. This section briefly examines some of the more popular protocols proposed..

2.1 Centralized Approaches

Some of the first approaches to appear followed the centralized client-server architecture. What all these models have in common is the reliance upon a centralized storage that would handle queries by users. This assumption violates the requirements of ad hoc networks where all nodes should be considered equal and no one node should be given extra responsibility when compared to its peers. Legal problems have led to the downfall of most such centralized schemes. In this section we examine several paradigms that follow this architecture.

The rise and fall of the Napster service [1] is an example of such file sharing protocols. Other centralized approaches are discussed further in the thesis when considering the specific problem of service discovery. The protocol used by Napster before its demise is a classic client-server approach. Users would connect to a centralized server database and upload a list of the files they host. When a user wants to find a file, he would query the server for the locations of users hosting that file. He would then select one of these hosts and establish a connection for the transfer of the file itself.

A particular type of content that has a large impact on computing is services. A service may be any software or hardware that may be used by an application to accomplish some work.

Popular protocols in this field are the Service Location Protocol (*SLP*) [18], the Jini Lookup Service [13, 14], and the Ronin Agent Framework [14, 19, 20, 13]. Like other centralized approaches, these too depend on a central directory server. Their respective algorithms are similar. Nodes hosting services register them with the central directory server. A client that is looking for a particular service contacts the directory server in order to resolve its needs. Once the client knows what services are available on the network, it contacts the respective server directly in order to accomplish the desired task.

Unfortunately for for most such centralized approaches, this setup makes it easy to isolate a responsible entity in case litigation is possible. Such litigation resulted in the demise of Napster as a commercial enterprise and, along with the desire to reduce dependency on a central point of failure, has pushed the need for decentralized approaches which have come to be known as peer-to-peer.

2.2 Decentralized (Peer-to-Peer) Approaches

The term peer-to-peer was originally used to describe any scheme that allowed users to share files directly with each other, including such centralized approaches as those described above. However, with lower interest in such protocols for sharing files, the term has shifted in meaning. Currently it is used to describe only decentralized approaches that do not require a central database for content location and each node in the distributed system has identical capabilities and responsibilities. This section focuses on such decentralized schemes.

2.2.1 Content Location in Resource-Rich Networks

Peer-to-peer has gained momentum in large due to the availability of resource-rich networks. In such an environment, a prevailing idea is the use of hash tables for locating content. However, to remove the reliance on a central server, the hash table is spread among nodes in the network. A number of approaches have been proposed that have gained acceptance in this field. Such algorithms are *CAN* , *Chord*, *Pastry*, *Tapestry* and other systems based on them.

The Content-Addressable Network (*CAN*)is proposed in [5]. Each node is assigned a key. The collection of keys makes up the coordinate namespace of the system. Coordinate space is partitioned among all nodes and every node holds a portion of the hash table called a *zone*. Each node would

know the addresses of the owners of its neighboring zones in the coordinate space. To insert a (*key*, *value*) object, a node hashes the key to a unique point, P, in the space. It then stores the pair in the node whose zone P is in. General routing and routing of requests is done greedily using the node *id*'s. When a node receives a request for an object, it hashes the object to get the key. It then forwards the request to its neighbor which is closest to the key's home zone. This continues until the request reaches the owner of the zone where the coordinate lies.

Chord [2] is another scheme that is similar to *CAN* in the case where a one-dimensional coordinate space is used. An *id* is generated for each node by using a hash function. The *id*'s for all nodes are ordered in a circular namespace. Each node knows the *id*'s of its neighbors in the circle. When a file or object is inserted into the network, it is hashed to produce a key. A key k is assigned to the first node whose *id* is equal to or follows k in the *id* circle. This node is called *successor* of k and will be the only node storing the (*key*, *value*) pair. Routing is done by forwarding messages along successive nodes in the circle until the successor of the required key is reached. A *finger table* is used to allow for jumps to distant points in the namespace in order to reduce the cost of traversal.

A scheme that puts the stress on protecting the anonymity of users is *Freenet* [6]. Similarly to the approaches considered previously, Freenet operates as a network of identical nodes that pool their collective storage space to store data and route queries. Similar to *CAN* and *Chord*, nodes and files are associated with keys. To insert a file, the user obtains a binary key for the file. He then sends an insert message to his node specifying the key and a hops-to-live value, which determines the number of nodes that will store the file initially. When a node receives the insert request it checks to see if the key is already taken. If there is such a key, it returns the file as if a request has been made to it. This signifies a collision to the user and he would have to use a different key. If there is no such file, it forwards the request to the nearest key in its routing table to the key specified by the insert message. This is repeated until the hops-to-live limit is reached or a collision returns the file. If a node receives a file in exchange to an insert request, it forwards it back to the user as if a request for the file had been made. If the hops-to-reach limit is reached without a collision being detected, an *"all clear"* message is propagated to the originator node. The original inserter then sends the data to insert which is propagated along the same path the initial insert

message set up. Each node along the path will store a copy of the data. Queries are forwarded identically to the nodes with key values closest to the key searched.

Pastry, [4, 21], is a distributed object location and routing scheme for peer-to-peer applications. Like the previous approaches, Pastry performs application-layer routing and object location. However, Pastry by itself is simply used as a routing substrate. On top of it a peer-to-peer system may be built. The functionality provided by Pastry is simple and limited. Each node in the Pastry network is assigned a unique identifier (*nodeId*). When queried for a numeric key, a Pasty node forwards the query to the node in its routing table with *nodeId* closest to the query key. While this is somewhat similar to *Freenet*, the two differ in the way they select the next hop. Pastry does not simply forward to the node with the closest key. Instead, it uses prefix lengths of the keys to make its decision. The node with the longest common prefix is chosen for the next hop. *PAST*, [21], is a large scale peer-to-peer utility that provides persistent storage of data built on top of the *Pastry*. PAST provides redundancy by storing an object not only on a single host but rather on several nodes with *nodeId*'s closest to the file key.

Tapestry, [3, 22, 23, 24], provides similar functionality to Pastry. It provides a mechanism for routing queries to a server holding a copy of a file. When a new object is to be inserted into the network, a method MapRoots is used to generate a set of root servers that will hold location information for the node serving the file. The server then sends messages towards all these root servers. This is done using the neighbor *id* closest to the root server. These messages contain the key to be inserted and the *id* of the server. The messages are propagated towards the root server and at each step create a backwards pointer to the node hosting the given key. Requests for objects are routed in a similar way. A node first generates the key for the object it is looking for and uses the MapRoots method to find the set of root servers. It then sends a query to one of the route servers. Similarly to inserting a node, the query is forwarded to the node with *id* closest to the root server. The query stops once a pointer is met along the way specifying the node hosting the file at which point it is forwarded directly to it. A query is guaranteed to find such a pointer in the root server in the worst case.

2.2.2 Content Location in Ad Hoc Networks

As the ad hoc network gained momentum, new protocols were required to handle content location. Some of the first proposals were based on the use of broadcast. There are two variations of this simple idea. On one hand, nodes hosting content periodically broadcast advertisements throughout the network. Such a mechanism assures that every node will know about every resource available in the network [15]. Thus, queries may be resolved immediately. This approach, however, leads to a prohibitive amount of proactive traffic in the network and wastes precious resources. On the other extreme, advertisements may be removed entirely and queries will have to be broadcast throughout the network [25], [17], [16]. This removes the proactive traffic but the cost of the queries grows substantially and more time will be required before queries are resolved. Such expensive queries may be prohibitive in an environment where frequent cooperation is required between nodes in the network. Such an environment may be a military operation where efficient and timely sharing of vital information is crucial to success.

As ad hoc networks matured, it became increasingly obvious that broadcast-based schemes are costly to deploy and that the traditional schemes such as the ones mentioned in Section 2.2.1 would not perform well under the new requirements due to their inability to take into account the scarce resource of the environment.

The Group-based Service Discovery protocol (*GSD*) [26] attempts to improve protocol overhead by eliminating broadcast queries completely. The protocol makes use of assigning services a hierarchical structure. Thus a specific printer may belong to the *"printers"* group, which in turn may be part of the *"output hardware"* group, etc. Nodes hosting services send periodic service advertisements to their neighboring nodes and to all nodes up to r hops away. The service advertisement message contains a field, *Other Services*, which enumerates groups of services that the node has seen in its vicinity. Upon receiving a service query, a node checks its table to see if the service is hosted by a node within r hops from it. If it is, the query is forwarded to that node. Otherwise, the *Other Services* field is used to select a set of nodes that have seen such services in their vicinity and may know a location hosting the requested service. The query is then forwarded to the selected nodes. At each step of the query, a backwards pointer is stored at the node similar to *AODV* [27]. If the query ultimately reaches a node hosting the service, the reply is sent back

along that path. *GSD* provides an relative improvement over protocols making use of broadcast. However, this is achieved at the cost of adding additional knowledge in the nodes as they must be aware of the entire hierarchical structure of services.

The idea to attempt to replicate distributed hash table functionalities, as present in *CAN* and *Chord*, is set forth in [28]. The proposed system consists of clients, Resource Providers (*RP*'s), and Discovery Agents (*DA*'s). Clients are regular hosts that may use resources and initiate content queries. *RP*'s are hosts that serve content for the clients. *DA*'s collectively maintain the distributed hash table of all available resources. *DA*'s periodically broadcast their addresses throughout the network. When a *RP* has a service to announce, it calculates a hash value that gives him the *DA* responsible for that part of the hash table. It then sends a content registration message to that *DA*. Clients send queries to the closest *DA*(*home DA*). If it is not aware of the existence of such a resource, the *home DA* computes the hash value of the requested resource and queries the appropriate *DA*. Such an approach does indeed provide some of the benefits of a distributed hash table as described in Section 2.2.1. However, in an ad hoc network, it wastes resources due to the periodic broadcasts to the entire network.

Another novel approach to efficient data dissemination in a dense ad hoc network is proposed in [29]. The protocol specifies that for each data source to be disseminated, a grid structure be established by the nodes in the network. Data will then be transmitted along members of the grid. Nodes on the inside of the so-formed cells will be bypassed by the data streams. A node that desires to receive data for a specific source need only flood a query inside its own individual cell. Any grid node that obtains a query will set up a pointer to the client and send data that it receives to the requesting client. However, certain shortcomings do exist with this scheme. The construction and maintenance of the grid for each individual data source is expensive. If there are just a small number of sinks in the network, the cost of establishing and maintaining a grid that encompasses the entire network is significant.

2.2.3 Trajectory-Based Routing

This section examines papers that deal with making use of geometric trajectories in content location in location-aware ad hoc networks. This work is of special importance because it provides the basis

for the protocol proposed in this thesis.

In [11], the authors propose the use of location information for use in routing. The protocol provides for all nodes to periodically send advertisements along geometric trajectories. At each node in the trajectory, a backwards pointer is set up establishing paths leading to the source host. Any node that wishes to communicate with another node need simply send a query along a path that intersects with the advertisement path. The query is then forwarded by the node on the path to the desired host. The host that receives a query may then send a reply to the requesting node.

This idea is further developed in [12]. The authors propose propagating the advertisements and queries in cross-shaped trajectories, thus guaranteeing two intersections. Queries are answered by nodes at the intersection of the advertising and query trajectories.

This is a simple and elegant approach that may be modified to work for a variety of resources available in a network. However, as the number of advertising servers grows, the amount of proactive traffic becomes prohibitive. Both of the above algorithms assume that each node will advertise a unique resource. This is not true, however, in many cases since duplicate content may well exist in a network. An example of such duplicate content may be several replicas of a file hosted by different servers or an identical service provided by several nodes in the network. Under these conditions, the above algorithms will not scale well since they do not take measures to limit the overhead for duplicate resources. Solving this scalability problem is a major contribution of this work.

2.3 Summary of Techniques

Each of the described approaches has associated benefits and shortcomings. All protocols have been suited best for work under some pre-defined assumptions. The centralized approaches described in Section 2.1 share common characteristics. Very few of them take link cost into account for locating content thus making them impractical for use in ad hoc networks. The only way proximity metric is enforced is the assumption that objects would register with servers close to them. Similarly, clients would query databases close to them first. The reliance on resource rich central nodes is typical of approaches based on regular networks. However, for ad hoc networks, this reliance is a violation of the principle of sharing responsibility equally among nodes.

The algorithms examined in Section 2.2 attempt to remove the reliance on a single point of

failure. Even though they have removed the reliance on a central node, most of them have created their own namespace and are entirely oblivious of the underlying network. This is again not a serious problem in a bandwidth-rich regular network. For ad hoc networks, such approaches are rather costly. *CAN*, *Chord*, and *Freenet* make insertion of nodes and object into the network dependent on the particular namespace used which is inefficient in resource-poor environments since the path taken may not necessarily be the shortest one. Some protocols were considered that attempt to alleviate these problems specifically for an ad hoc environment. However, even these schemes do not fare well as they are still wasteful of the limited resources available and do not scale well as the number of nodes on the network grows.

As we will see in Section 4, we integrate the techniques above in the design of GCLP along with our own insights. As such, we expect that GCLP will perform favorably as shown by good scalability and reliability of the provided service.

Chapter 3

Requirements for Proposed Protocol

Based on the examinations of the previous protocols made in Section 2 and their shortcomings, we require that our final GCLP protocol have the following properties:

- **Scalable**: Current protocols proposed for ad hoc networks do not scale well. Our protocol should be able to scale to hundreds or even thousands of nodes while still maintaining peak performance. The amount of control overhead needed to maintain resource availability information should scale nicely with an increase in resource availability.

- **Fault-tolerant**: It is quite possible that nodes can go down due to some unforeseen errors. As a result, our protocol cannot rely on a small subset of nodes for location information. Instead, this information should be available from multiple nodes and the information should be reasonably accurate. To avoid such possibility, it is important to provide certain redundancy of availability information.

- **Adaptable**: If our protocol were to be deployed in an ad hoc environment, it would have to support and adapt to node mobility. As such, it would have to be able to handle rapidly changing network topology without increasing cost and reducing reliability.

- **Balanced cost**: Since we attempt to work in an environment where communication between nodes and resource usage is high, we want queries to be relatively cheap while still allowing for lower overhead cost and scalability of the protocol.

- **Accurate**: We require that any query reply be answered only by nodes that have accurate information concerning the requested resource.

- **Efficient**: A response should specify the server that would result in the cheapest connection as measured by distance between the server and client. In a uniformly distributed dense network, a shorter distance would translate into smaller number of hops and thus smaller cost.

Chapter 4

Geography-based Content Location Protocol

In this section we provide the design description of *GCLP*. We start by giving a brief overview of the protocol before describing the details of the design

4.1 Protocol Overview

The proposed protocol treats all nodes in the network as equal. This means that the design is consistent with the principles ruling the distributed environment of a MANET. Namely, no single node takes more responsibility than others. Nodes may assume any of the following roles, more than one if required. A Content Server (*CS*) is a node that hosts one or more resources that may be used by other nodes on the network. Such nodes are responsible for advertising their hosted resources to the rest of the network. Content Location Servers (*CLS*'s) are nodes that host location information about one or more resources available. These nodes are responsible for providing timely and efficient responses to queries about specific content. *Clients* are nodes that request resources on behalf of an application or any other higher layer.

The basic protocol follows the scheme described in [12]. Periodically, a *CS* will transmit update messages (*UM*) to specific nodes in the network. These updates advertise available resources and the *CS* that hosts them. *UM*'s follow a predefined trajectory through the network similar to the trajectory-based schemes described in Section 2.2.3 [11, 12]. This significantly decreases the amount of proactive traffic as it is limited to nodes along the trajectories. Nodes along these trajectories cache the information received from the updates. A node that stores such information becomes a

CLS. If it receives a query about content it knows the server of, it will reply with the server address.

A client may locate any content on the network by sending out a query message (*QM*). The query is similarly propagated along predefined trajectories. In a dense network, these trajectories are guaranteed to intersect at least one update trajectory. The *CLS* at the intersection point that receives the query responds with a reply message (*RM*) that is sent back to the client. Upon receipt of a reply, the client may establish a direct connection with the content server using the underlying routing protocol to make use of the available resource. The queries follow a forwarding scheme that keeps their cost low while finding the closest available server in the vast majority of situations. Finding the closest *CS* available is an important benefit as it generally means fewer hops between the client and the server, which, in turn, translates into lower connection cost.

To make sure that all nodes on the network know the location of all their neighbors when selecting next hop hosts for the updates and queries, a third type of message is used. The *hello* message is periodically broadcast by each node on the network to its one-hop neighbors to advertise the node's position.

The following sections explain the functionality of all three types of nodes in detail. Proofs are provided to demonstrate the correctness of the protocol and its ability to keep cost low.

4.2 Protocol Details

This section provides details about the design of *GCLP*. It describes the three phases of the protocol – neighbor discovery phase, content advertisement, and content discovery. It also shows proof of protocol correctness and efficiency.

4.2.1 Neighbor Discovery

The neighbor discovery phase allows nodes to learn the location of their one-hop neighbors. This is achieved by the use of beaconing. Each node periodically broadcasts a Hello Message (*HM*) with a time-to-live of one. This assures that all of its immediate neighbors receive the message. The format of the *HM* is given in Table 4.1. As shown in the table, the format of a *HM* is simple and serves the single purpose of advertising a node's current location.

Field	Size
Message Type	4 bits
Source Address	4 bytes
Source Location	4 bytes

Table 4.1: Format of a Hello Message (*HM*)

Upon receiving a *HM*, a node updates its *neighbors table* with the new information. This insures that all nodes up-to-date and accurate information about their neighbors' geographical locations.

4.2.2 Content Advertisement

Content advertisement is performed by periodically sending Update Messages (*UM*) through the network, similar to [12]. Each *CS* periodically initiates advertisements by sending *UM*'s in four opposite geographical directions, or trajectories–North, South, East, and West. Each advertisement specifies the location of a single resource on the network. To accomplish this, the *CS* uses the geographic location of its immediate neighbors to select the next *CLS* in the given direction. The algorithm for selecting next-hop nodes is described in detail below. The *UM* is then sent to that node. Upon receiving the update, the chosen *CLS* adds the information to its *Content Location Table* and uses the same algorithm to decide which node in the direction of the update will be the next *CLS*. This continues until a node on the fringe of the network discovers that there are no neighbors in the sector along the update's direction.

This basic advertising algorithm is exemplified in Figure 4.1. Here, update messages are being propagated for a particular content server through the network in the four directions. The nodes that are chosen to become *CLS*'s are colored.

We next examine the format of an *Update Message*. It is given in Table 4.2. The first four bits specify the format of the message as a *Update Message*. The *direction flags* specify which geographical direction this message is sent in. These are used to decide which sector to locate the next hop node in. The *next hop* is the address of the selected *CLS*. The *Content ID* identifies the resource that is being advertised. The server address and location specify the content server and its geographical location. The geographical location of the CS may be used to establish a connection between the client and the server in case a geographic location-based routing algorithm is used.

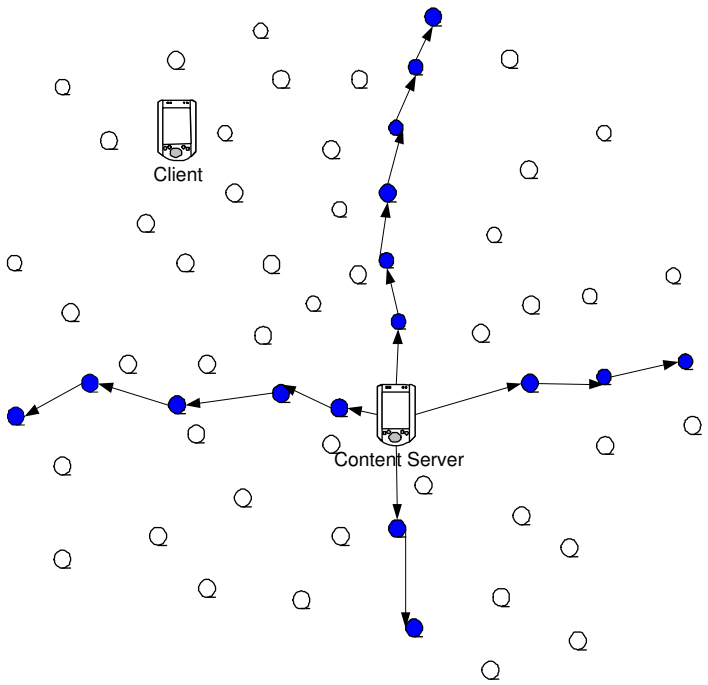

Figure 4.1: A Content Server propagating updates through the network. The server's selected content location servers are colored.

Field	Size
Message Type	4 bits
Direction Flags	4 bits
Next Hop	4 bytes
Content ID	4 bytes
Server Address	4 bytes
Server Location	4 bytes

Table 4.2: Format of a Update Message (UM)

An important part of the protocol is the selection of the CLS nodes along the geographic direction of a UM. In designing an algorithm several heuristics may be used:

- Picking nodes closest to the line of the direction. This keeps the line of CLS's straighter but may not be as efficient since some CLS's may be needlessly close to each other even if nodes are available farther away in the sector.

- Picking nodes with greatest distance from the current node. This may be more efficient as it provides for greater spaces between neighboring CLS's. However, this approach may lead to update trajectories that are not straight.

- A combination of the two. This is the approach that we take and we describe the exact algorithm below.

In [12], the authors propose using the first two for selecting next hop nodes in the update path–select nodes farthest away from the current node or select nodes closest to the trajectory line. We modify this basic selection algorithm as follows. When selecting next hop *CLS*'s, we would like to achieve two things–cover a larger distance between *CLS*'s and keep the update trajectory as straight as possible. To accomplish this, each node in the 90° sector along the trajectory is assigned a rating based on Equation 4.1, where R is the rating for the given node in the sector, d is the distance from the node making the decision, and r is the offset from the geographical direction line. This algorithm allows for nodes farther away and closer to the perfect trajectory to have highest ratings.

$$R = d/r \tag{4.1}$$

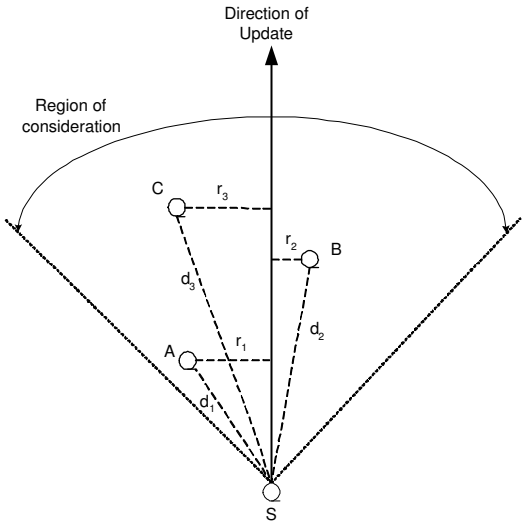

Figure 4.2: A node picks the next hop for an update message among nodes in the appropriate sector.

This is further clarified by Figure 4.2. A node, S, is considering three possible candidates in the desired sector, A, B, and C. Based on the formula provided, node B will be selected as the next hop in the trajectory as it will have the highest rating.

The selection of 90° as the sector size is dictated by the need to increase node availability in

each sector given sparse networks. Smaller sector sizes would not provide any improvement since the algorithm proposed will pick nodes closer to the trajectory no matter the sector angle. Thus, we can only benefit from allowing more nodes to be considered for selecting next hop neighbors in the update and query trajectories. If a node cannot find a neighbor in a given sector that it is trying to transmit to, the trajectory in the given direction is interrupted. In Section 7.2, we examine an alternative to this heuristic.

As described until now, the protocol is simply an implementation of [12] and does not scale well. It does not deal with the possibility of more than one server hosting the same resource in the network. Next, we describe a major component of our protocol that provides for protocol scalability and cost efficiency. If *UM*'s for duplicate resources are allowed to propagate throughout the network, the resulting traffic will be overwhelming. Instead, each *CSL* chooses whether to forward a *UM* or not. If a *CSL* receives multiple advertisements for a particular resource, it will only forward updates from the *CS* closest to it. In case there is a tie, the *CLS* will continue to propagate updates from the first server it received advertisements from. The resulting advertisement grid allows for scalability of the protocol as each additional replica of a resource will introduce less and less proactive traffic into the network. An example is shown in Figure 4.3. It also has the important property that each *CSL* will know the location of servers closest to them and thus answer queries resulting in data connections of smallest cost possible as measured by the number of hops.

4.2.3 Content Discovery

To locate content on the network, a client sends out a query message through the network in a manner identical to the *UM*'s. A query is sent in the four geographical directions. The next hop in the query trajectory is selected using the same algorithm described in Section 4.2.2. The format of the query message (QM) is shown in Table 4.3. The *message type* identifies the packet as a query message. The *direction flags* specify the direction along which the message is to be propagated. The *next hop* identifies the chosen next hop in the trajectory. The *Content ID* is identifies the resource the query is attempting to discover. The *requestor address* and *location* identify the node sending out the query and its geographical location.

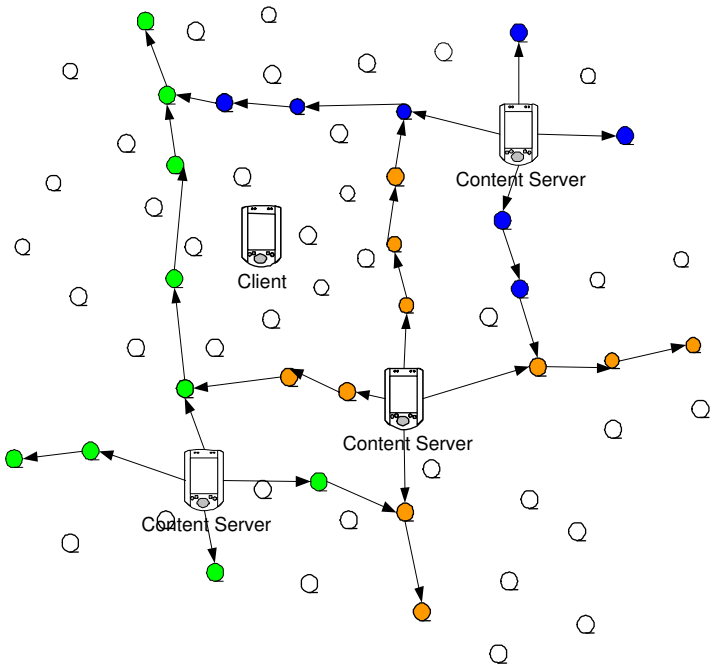

Figure 4.3: Several servers advertise availability of the same resource. The content location servers for each server are given in different colors.

Field	Size
Message Type	4 bits
Direction Flags	4 bits
Next Hop	4 bytes
Content ID	4 bytes
Requestor Address	4 bytes
Requestor Location	4 bytes

Table 4.3: Format of a Query Message (QM)

A *content location server* that receives a QM will send a query response message (RM) to the requestor. The RM follows regular greedy geographic routing with each node forwarding the packet to its neighbor closest to the destination. The format of a response message is given in Table 4.4. The *message type* identifies the message as a *query response*. The *next hop* is the address of the next node on the path back to the *client*. The *Content ID* identifies the requested resource. The *server address* and *location* specify the content server hosting the resource and its position. The *requestor address* and *location* identify the client querying for content and its location.

The content discovery process is exemplified in Figure 4.4. Here a client sends out queries through the network in four directions. Once the queries reach a *CLS* along the update trajectories,

Field	Size
Message Type	4 bits
Next Hop	4 bytes
Content ID	4 bytes
Server Address	4 bytes
Server Location	4 bytes
Requestor Address	4 bytes
Requestor Location	4 bytes

Table 4.4: Format of a Response Message (RM)

it answers with a response message. The client may receive more than one response messages to the same query. In such a case, it picks the response identifying a CS closest to its geographical location.

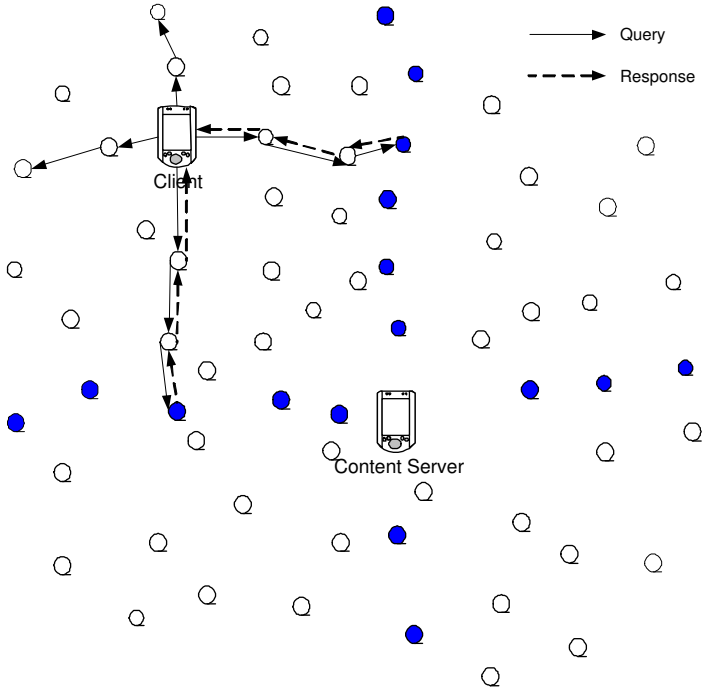

Figure 4.4: A client attempts to locate content by sending queries through the network. The queries are answered once they reach a content location server.

Chapter 5

Analysis of GCLP

In this section we provide an analysis of the Geography-based Content Location Protocol, including proof of correctness. It is important to point out that the network under consideration in this section is a dense network with nodes at each point in the logical space. Also, in the context of this discussion and of a dense network, a hop is assumed to mean one transmission range.

5.1 Proof of Correctness

To prove that at least one intersection of update trajectories and query trajectories still exists, even with some update trajectories being interrupted by updates from closer *CS*'s, we need to prove that for at least one trajectory will propagate in each one of the four geographical directions. The proof is simple and is illustrated in Figure 5.1. We have two *CS*'s, *S1* and *S2*. Two *CLS*'s, *A* and *B*, interrupt some update trajectories and forward others. To prove that at least one trajectory is propagated in each direction, let us observe what happens at a point where a trajectory is interrupted, such as point *B* in the figure. It is sufficient to observe that in order from one trajectory to get interrupted, such as the trajectory form *S1*, there must exist a server whose perpendicular trajectory causes the interruption at that point, *S2*. Such a server would be propagating its own updates in the direction of the interrupted trajectory. Thus, there is at least one update trajectory in each direction.

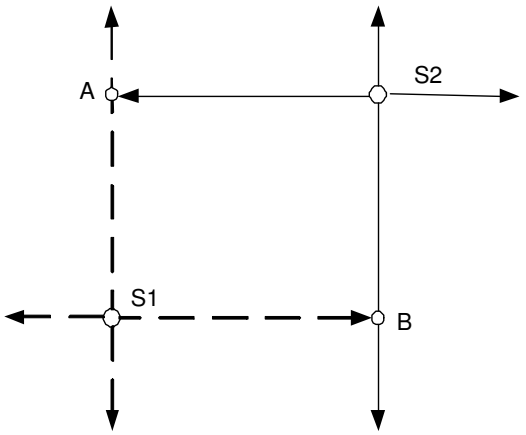

Figure 5.1: At least one trajectory will propagate in the direction of an interrupted update trajectory.

5.2 Scalability Analysis

To provide analytical model of our protocol, we examine the performance of the protocol in the two extreme cases–when there is a single server in the network and when all nodes in the network are servers. We then consider how the proactive traffic introduced by update messages changes with growing the number of servers in between the two extremes.

In the first case, we have a single content server in the network for a given resource. Consider a network of dimensions w by l hops. Any update sent by the *CS* in the four directions will have a total cost of $w+l$. Thus, in the case of a single *CS*, the cost is *O(w+l)*.

In the other extreme, consider the case where all nodes in the network host the same content. In this case, the updates from each host will be ignored by their next hop *CLS*'s because they already host the resource. Thus, the cost of each individual update is *O(1)*, giving a total cost in the network of *O(n)*, where n is the number of *CS*'s in the network.

To show the complexity of the protocol between the two extreme cases, we consider the proactive traffic generated by adding additional servers. Let us assume that the first server, *S1*, is located centrally to a network that occupies a square-shaped region. Such a central placement of servers minimizes the search cost for a given number of servers since all queries have an equal chance of traversing the same size regions. If the server is not centered, then queries will have a greater possibility of originating from and traversing the larger and thus most expensive regions. *S1*

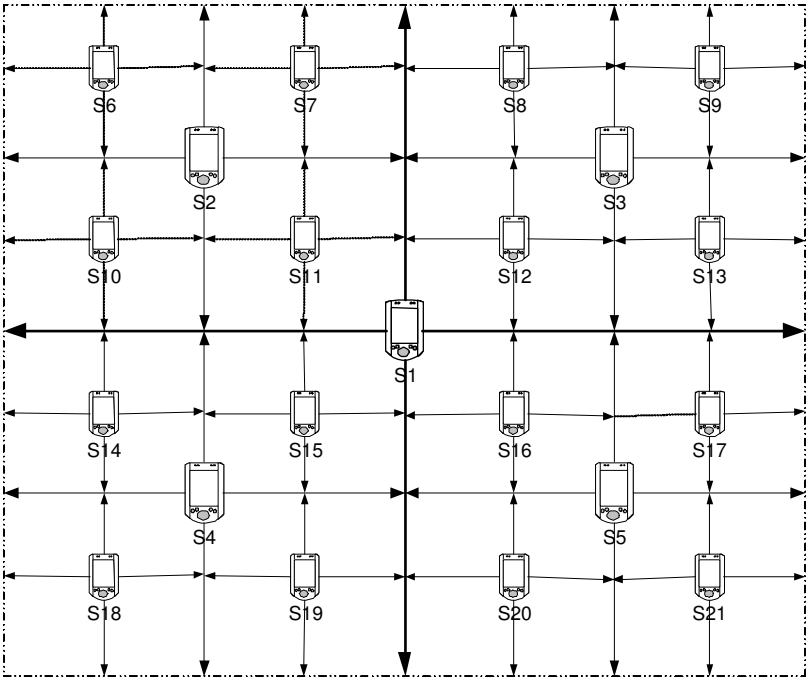

Figure 5.2: A uniform distribution of servers in a network. *S1* is the central server. *S2-S5* are the second-tier servers. *S6-S21* are third-tier servers.

generates update trajectories that will go until the edge of the network, producing two complete rays. A *ray* is considered to mean a group of update trajectories that span the network from end to end once. This is illustrated in Figure 5.2. Let us now add four new servers, which we call second-tier servers, one in each of the four quadrants around *S1*. The proactive traffic from each one of these servers is limited to the quadrant that hosts the corresponding server. Thus, the four second-tier servers generate at most a total of four rays. This means that each of the four new servers contributes a total of one ray per server. This exercise may be continued by placing 16 new third-tier servers in each square produced by placing the previous four servers. Traffic from each of the 16 new servers will be limited to the small square resulting from the updates traffic from the previously placed servers. It is easy to see that the new 16 servers will produce eight new rays. Thus, each of them produces only a *1/2* ray per server. This series is shown in Table 5.1.

The above series yields the following algebraic series that describes the proactive traffic in the network based on the number of servers.

Servers	Rays	Rays per Server
1	2	2
4	4	1
16	8	1/2
64	16	1/4

Table 5.1: New overhead traffic introduced by additional servers.

$$\textit{Rays per Server} = 1(2) + 4(1) + 16(1/2) + 64(1/4) + ... \tag{5.1}$$

$$\textit{Rays per Server} = \frac{\sum_{i=0}^{k}(4^i \times 2/2^i)}{\sum_{i=0}^{k} 4^i} \tag{5.2}$$

$$\textit{Rays per Server} = \frac{2 \times 2^{k+1} - 2}{\frac{4^{k+1}-1}{3}} \tag{5.3}$$

$$\textit{Rays per Server} = \frac{6(2^{k+1} - 1)}{(2^{k+1} - 1)(2^{k+1} + 1)} \tag{5.4}$$

$$\textit{Rays per Server} = \frac{6}{2^{k+1} + 1} \tag{5.5}$$

From Equation 5.5 it can be observed that protocol overhead generated per server drops exponentially as new servers are added to the network. This uniform distribution of servers is also optimal for search cost since all queries will only be propagated within a small square. Graphing Equation 5.5, we observe the pattern that we expect to find in our simulations when considering overhead traffic added by introducing new servers into the network. This is shown in Figure 5.3. Notice that, as mentioned above, for large numbers of servers the traffic introduced by new servers will be constant and the complexity of the algorithm approaches $O(n)$.

Uniform distributions of nodes like the one considered above, however, may be hard to achieve under real-world conditions. Let us consider what happens when disorder is introduced into the system. We assume that one of the four second-tier servers introduced moves outside its square. This is illustrated in Figure 5.4. When it enters a new square, its update traffic will be limited to the new home square. This new square will be smaller as it already hosts a server. Thus, the

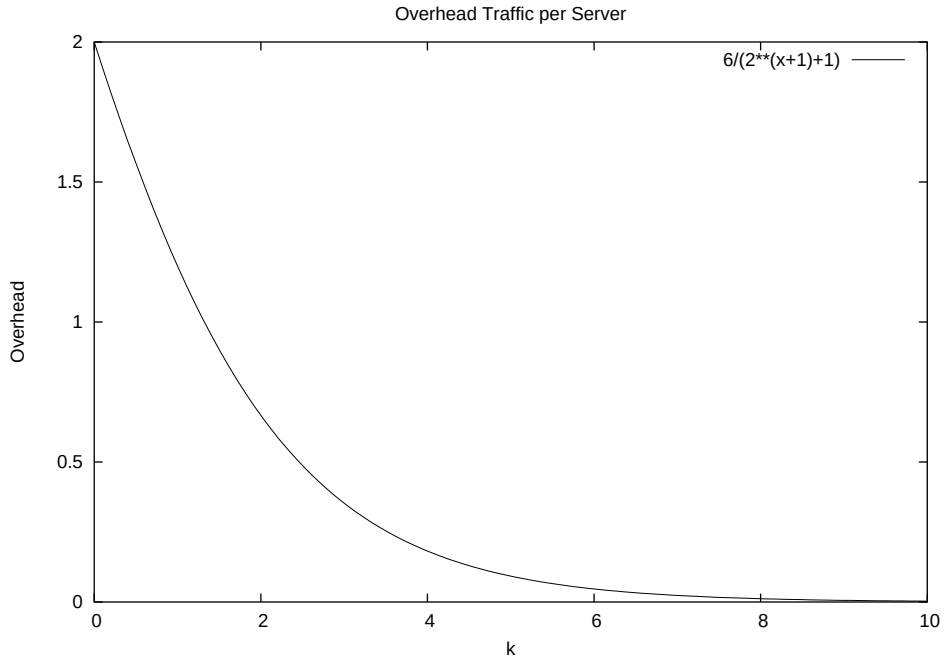

Figure 5.3: Expected pattern of overhead traffic per server as a function of number servers.

proactive traffic generated by additional servers is smaller if they are not uniformly distributed. On the other hand, the cost of queries grows as all the nodes from the home square deserted by the servers will produce on average more expensive queries.

Our simulations show that the overhead traffic of the protocol does indeed scale well and additional servers lead to exponential drops in new overhead traffic. The scalability of the protocol is clearly seen in the simulations presented in Section 6.

5.3 Response Analysis

An desired property for the proposed protocol is the ability of queries to locate the closest content server available. However, under certain conditions, this may not be possible. This anomaly is shown in Figure 5.5. There are two servers in this setup, *S1* and *S2*. They propagate their updates according to the protocol described in Section 4.2.2. A CLS, *CLS1*, at the intersection of the two trajectories from *S1* and *S2* will only propagate updates from the server closer to it. In the figure, this means that only updates from *S1* will be propagated by *CLS1*.

Let us consider the different possible positions of a client in this environment defined by the

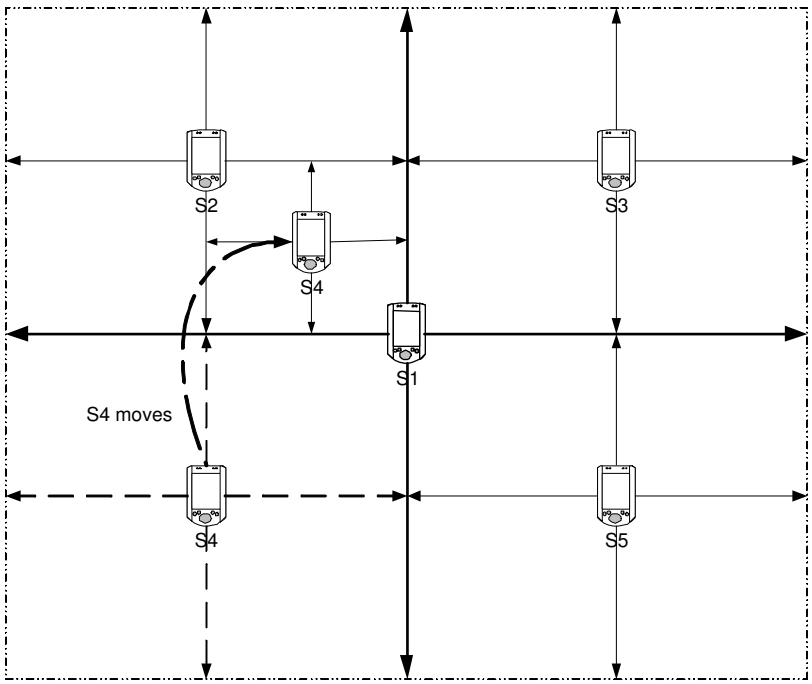

Figure 5.4: A second-tier server, *S4*, moves to a new location in the network. Update traffic that ceases with the relocation of the server is marked in dashed lines.

update trajectories from *S1* and *S2* and the line, l, which defines the set of points of equal distance to both servers. If a client is on the side of l where *S1* is located, then *S1* is the closest server to that client. Queries sent in that sector will intersect at least one update trajectory from S1 thus resulting in discovering the closest server. In this case, the closest server is discovered.

If a client is located on the side of l towards *S2* and not between l and the downwards update trajectory from *S1* (i.e., not in the region R in the figure), then it is closest to *S2* and queries from such a client will intersect at least one update trajectory from *S2*, thus again finding the closest server.

The problem occurs when a client, Q, is located in the region R bounded by the line l and the downwards update trajectory from *S1*. In this region, *S2* is the closest server. However, a query will be answered by a location server, $CLS2$, positioned on the downwards trajectory from *S1* resulting in locating a server that is not the closest one, *S1* instead of *S2*. Notice, that a similar region exists, R' where $S1$ is the closest server but only $S2$ can be found. In this section we prove that even in this situation, in the worst case, the error is relatively small when compared with the

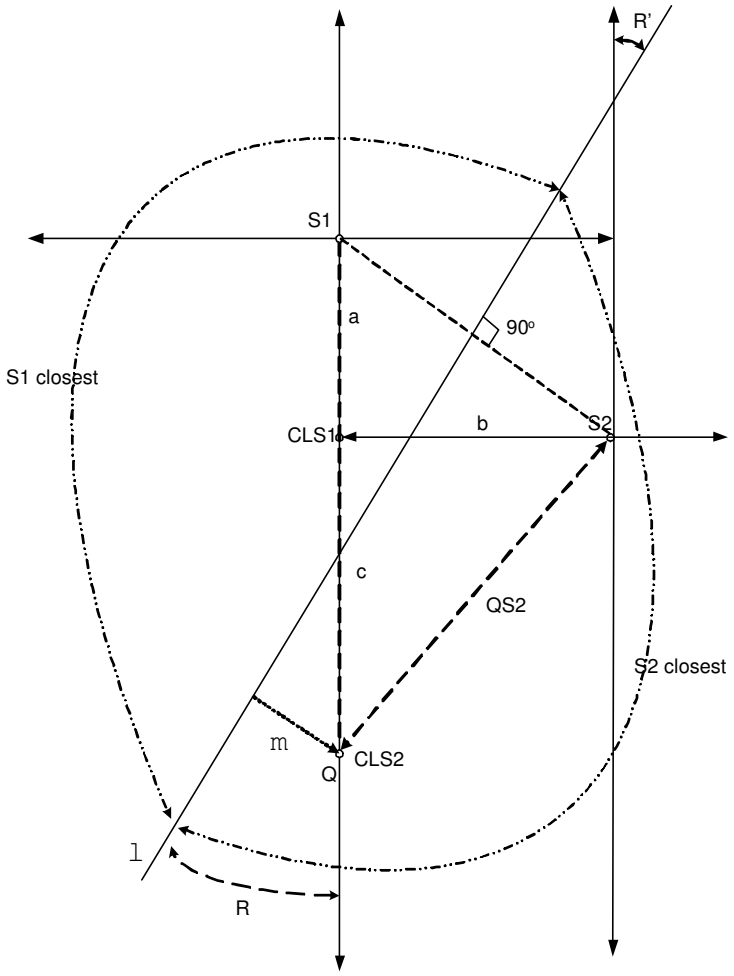

Figure 5.5: A client, Q, may locate a resource on a more distant server, QS1, instead of the closer QS2.

actual distance to the closest content server. This error is measured as the ratio *QS1/QS2*, where *QS1* is the distance between Q and *S1* and *QS2* is the distance between Q and *S2*.

First, let us examine how the motion of Q along a line m perpendicular to the line l affects the ratio. It is easy to see that any move of Q farther from l is moving it farther away from *S1* and closer to *S2*, increasing the ratio *QS1/QS2*. Thus, as Q is closer to the update trajectory, the ratio will be larger, giving us a greater error. For the purposes of the following analysis, we assume the worst case where Q is actually on the trajectory line itself.

Let us express the ratio of *QS1/QS2* in terms of the distances a, b, and c, as shown in Figure 5.5. Equation 5.6 shows this formula.

$$QS1/QS2 = (a + c)/(\sqrt{b^2 + c^2}) \tag{5.6}$$

To find the worst case, we take the derivative of *QS1/QS2* with respect to the distance c and see where it is *0* (Equation 5.7).

$$d(QS1/QS2)/dc = 0 \tag{5.7}$$

$$d((a + c)/(\sqrt{b^2 + c^2}))/dc = 0 \tag{5.8}$$

$$(b^2 - c \times a)/(b^2 + c^2)^{3/2} = 0 \tag{5.9}$$

Solving Equation 5.8 for c gives us a ratio of $c = b^2/a$. To prove that this is indeed a worst case, we use the second derivative with respect to c:

$$d^2(QS1/QS2)/dc = d((b^2 - c \times a)/(b^2 + c^2)^{3/2})/dc \tag{5.10}$$

$$= (2ac^2 - 3b^2c - ab^2)/(b^2 + c^2)^{5/2} \tag{5.11}$$

$$= -\frac{a^4}{b^3} \times \frac{1}{(a^2 + b^2)^{3/2}} < 0, \text{ for } c = b^2/a \tag{5.12}$$

Thus, $c = b^2/a$ achieves the maximum for $QS1/QS2$. Plugging back into Equation 5.6, we get a worst case ratio as follows:

$$QS1/QS2 = (a + b^2/a)/(\sqrt{b^2 + (b^2/a)^2}) \tag{5.13}$$

$$= ((a^2 + b^2)/a) \times a/(b\sqrt{a^2 + b^2}) \tag{5.14}$$

$$\Rightarrow QS1/QS2 = \sqrt{a^2 + b^2}/b \tag{5.15}$$

We can observe that for $CLS1$ to propagate updates from $S1$ and not $S2$, we must have $a \leq b$. This leads to the following conclusion:

$$QS1/QS2 = \sqrt{a^2 + b^2}/b \leq \sqrt{2b^2}/b = \sqrt{2} \tag{5.16}$$

According to Equation 5.16 we can conclude that, in the worst case, *QS1/QS2* $\leq \sqrt{2}$. Thus, the maximum error produced by our protocol as measured by the ratio between the distance found and the actual closest distance is relatively small, $\sqrt{2}$ times the optimal in a dense network.

Chapter 6

Simulation Results

In this section we evaluate the performance of our protocol in a simulated environment. The *ns-2* simulator was used to simulate a variety of network conditions. The area over which the simulated network was situated was 2000 by 2000 meters. A variety of network densities were simulated. For each case, 20 simulations were run and the results were averaged to produce the presented data. A sparse network of 50 nodes, moderate density networks of 100 and 200 nodes, and a dense network of 500 nodes were simulated. Static topologies are presented for networks of all densities. To study the effects of mobility, a moderate density network of 100 nodes was evaluated with varying nodes speeds. The Random Waypoint mobility model was used to simulate node mobility. Transmission range of the wireless nodes used for the simulations was 250 meters.

Several measures were considered when evaluating the protocol. The proactive traffic due to Update Messages was a major factor to the scalability and adaptability of our *GCLP*. The cost and success rate of queries is another crucial piece of information. To evaluate these properties, the following measures are considered in evaluating protocol performance:

- **Updates per Initialization**: This measure takes into account total proactive traffic in the network for each initialization cycle. Nodes initiate updates every 30 seconds. This measure is derived by accounting for all transmissions of *UM*'s during the run of the simulation and dividing by the number of times each node has initiated an update. This measure shows how protocol overhead varies as a function of the number of servers in the network.

$$Updates\ per\ Initialization = \frac{Total\ UM's\ Transmitted}{Number\ of\ Update\ Initializations}$$

- **Updates per Initialization per Server**: This measure is computed by dividing the *Updates per Initialization* by the number of servers present in the network. This metric is directly related to the *Rays per Server* metric discussed in Section 5.2 since the cost of the ray is directly proportional to the number of hops in the network and thus to the number of update messages necessary.

$$Updates\ per\ Initialization\ per\ Server = \frac{Updates\ per\ Initialization}{Number\ of\ Servers}$$

- **Hops per Query**: A goal of the protocol is to keep query cost low. This measure is used to quantify the cost of searching the network for content. In our simulations, each node in the network initiates a query once. The Hops per Query is calculated by taking the number of all transmissions of *QM*'s and dividing by the number of queries initiated.

$$Hops\ per\ Query = \frac{Total\ Query\ Messages}{Number\ of\ Queries\ Initiated}$$

- **Success Rate**: The accuracy of our protocol is measured by this metric. It is computed by taking into account the number of queries that receive responses and dividing by the total number of queries. If a query receives more than one response, it is only counted as successful once.

$$Success\ Rate = \frac{Successful\ Queries}{Total\ Number\ of\ Queries\ Initiated}$$

These results of the simulations, as measured by the above metrics, are described in the following sections.

6.1 Effects of Network Density

This section studies the effects of node density in the network on the performance of *GCLP* as measured by the factors specified in Section 6.

6.1.1 GCLP in Moderate-Density Networks

As shown in Figures 6.1 and 6.2, the total overhead traffic introduced by the update messages grows faster for small numbers of servers and the growth slows with the addition of servers. This follows a

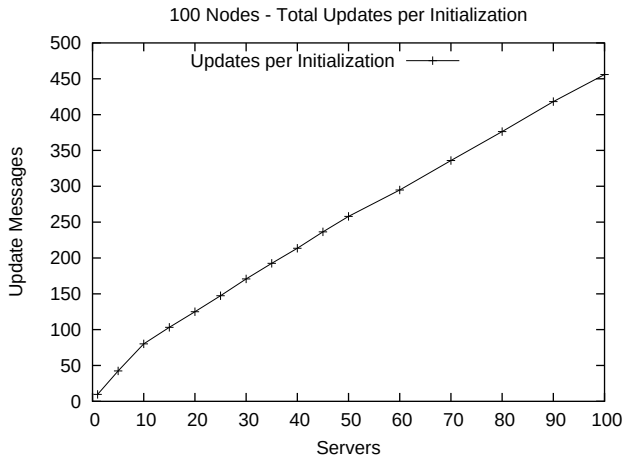

Figure 6.1: Proactive Update traffic in a network of 100 nodes as a function of number of servers.

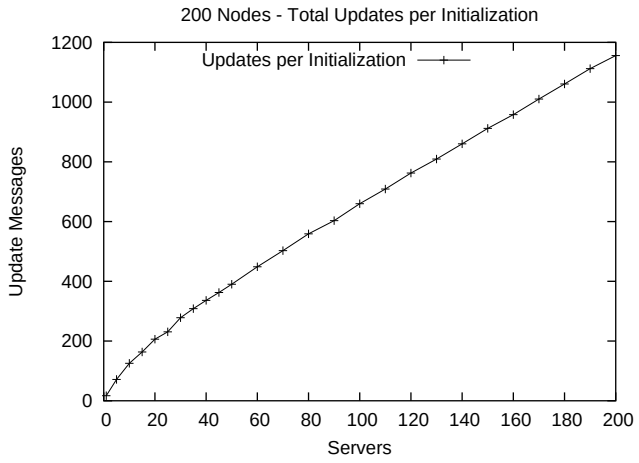

Figure 6.2: Proactive Update traffic in a network of 200 nodes as a function of number of servers.

pattern described in Section 5.2. As the number of servers grows large, the growth slows down and eventually becomes virtually linear. As seen in Figures 6.3 and 6.4, the amount of update messages per server shrinks rapidly with the addition of new servers. This pattern relates directly to the analysis in Section 5.2. The quick drop can be seen in update traffic per server with the addition of servers, as well as the constant complexity for large numbers of servers. This is due to the construction of the update grid. The above mentioned figures clearly demonstrate the scalability of the protocol.

The amount of query traffic also decreases rapidly with the addition of servers. This is to be

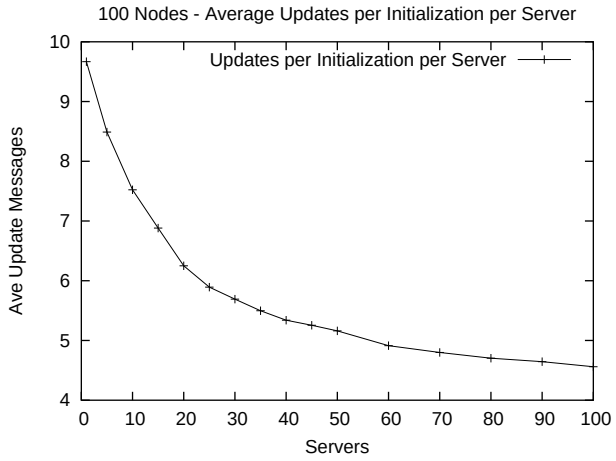

Figure 6.3: Proactive Update traffic per server in a network of 100 nodes as a function of number of servers.

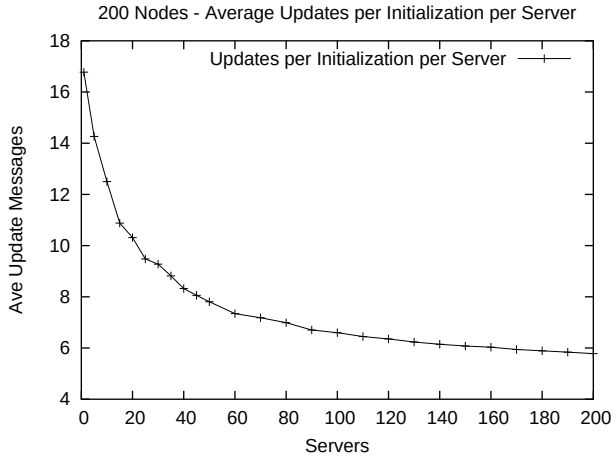

Figure 6.4: Proactive Update traffic per server in a network of 200 nodes as a function of number of servers.

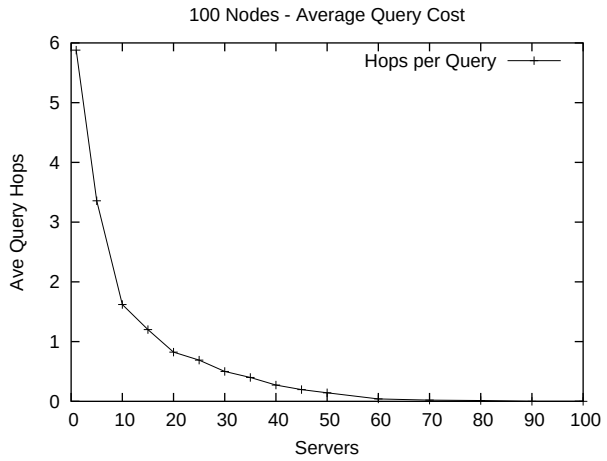

Figure 6.5: Query messages generated per application layer query in a network of 100 nodes.

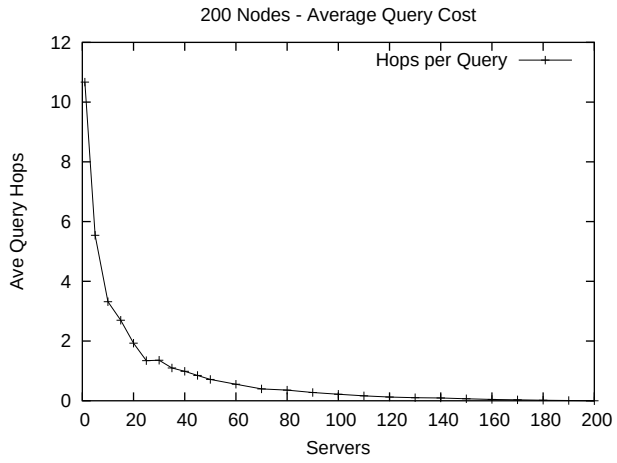

Figure 6.6: Query messages generated per application layer query in a network of 200 nodes.

expected as the grid grows denser and more nodes have the ability to answer queries. This property is shown in Figures 6.5 and 6.6.

The success rate for queries is virtually 100 percent in many cases. This is not true however for the cases of small numbers of servers and particularly in the less dense 100-node network. We stipulate that the reason is the non-perfect connectivity in the not too dense network. However, as redundancy is introduced with the addition of new servers, the success rate quickly grows to reach a virtual 100 percent. These tendencies are shown in Figures 6.7 and 6.8.

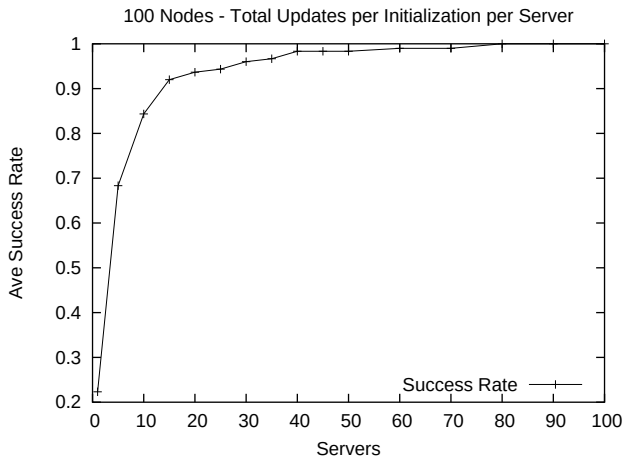

Figure 6.7: Query success rate in a network of 100 nodes as a function of the number of servers available on the network.

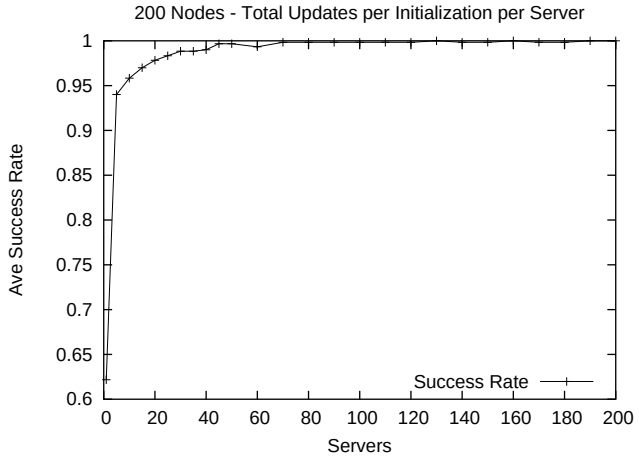

Figure 6.8: Query success rate in a network of 200 nodes as a function of the number of servers available on the network.

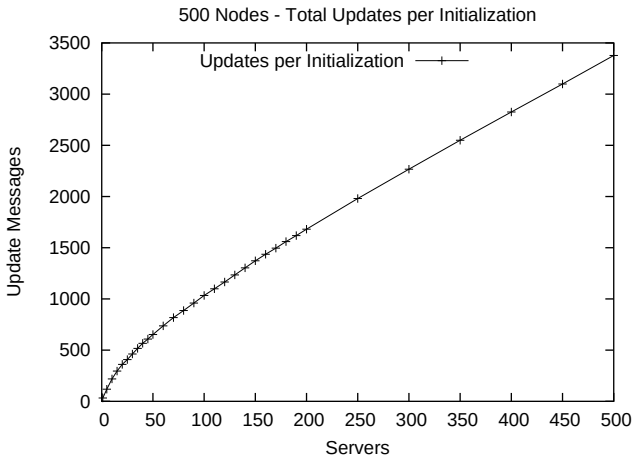

Figure 6.9: Proactive Update traffic in a network of 500 nodes as a function of number of servers.

6.1.2 GCLP in High-Density Networks

To simulate a high-density network, 500 nodes were placed on the same area of 2000 by 2000 meters. In this environment, the shape of the *Updates per Initialization* curve in Figure 6.9 clearly shows the larger growth for small numbers of servers and the nearly linear growth for higher server densities. This observation is further conveyed in Figure 6.10. The figure shows the good scalability property of the protocol as a growing number of servers rapidly leads to less proactive traffic per server and, eventually, a constant overhead traffic per server for large numbers of servers as expected from Section 5.2. This is due to the construction of the update grid.

The cost of queries also decreases quickly with growing numbers of servers. This is shown in Figure 6.11. At the same time, the success rate is virtually 100 percent for any number of servers, as shown in Figure 6.12. This is due to the good connectivity of the network and its high density.

6.1.3 GCLP in Sparse Networks

The performance of *GCLP* in sparse networks is evaluated next. Figures 6.13 and 6.14 show that proactive traffic still grows with the number of servers. However, the proactive traffic per server does not seem to vary significantly with the number of servers. This is due to the low connectivity of the network. In such an environment the performance of the protocol is lowered by the inability of nodes to establish connections with each other due to limitations of their transmission range.

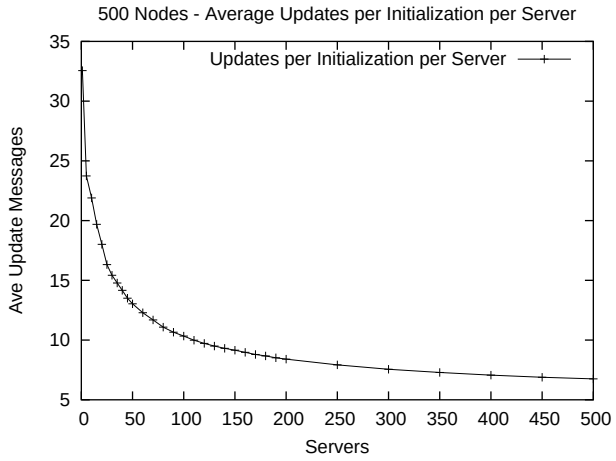

Figure 6.10: Proactive Update traffic per server in a network of 500 nodes as a function of number of servers.

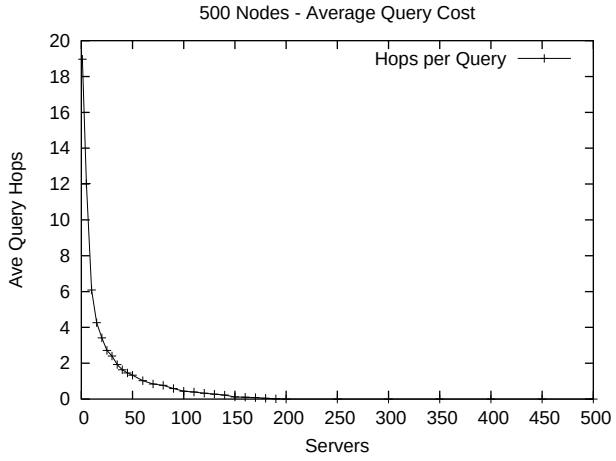

Figure 6.11: Query messages generated per application layer query in a network of 500 nodes.

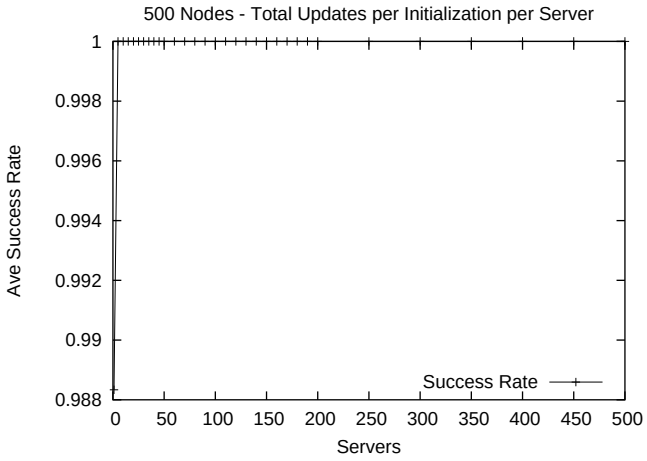

Figure 6.12: Query success rate in a network of 500 nodes as a function of the number of servers available on the network.

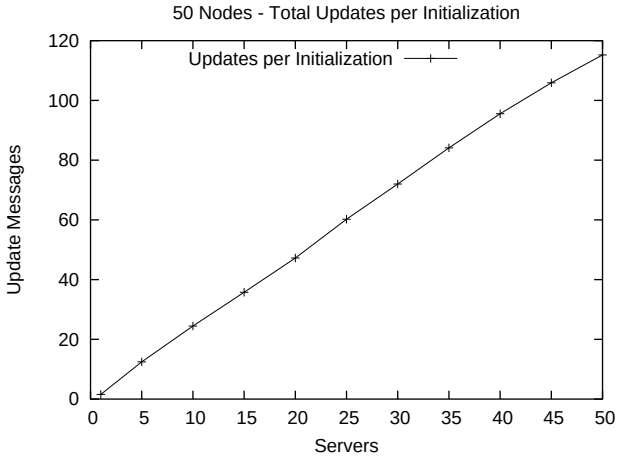

Figure 6.13: Proactive Update traffic in a network of 50 nodes as a function of number of servers.

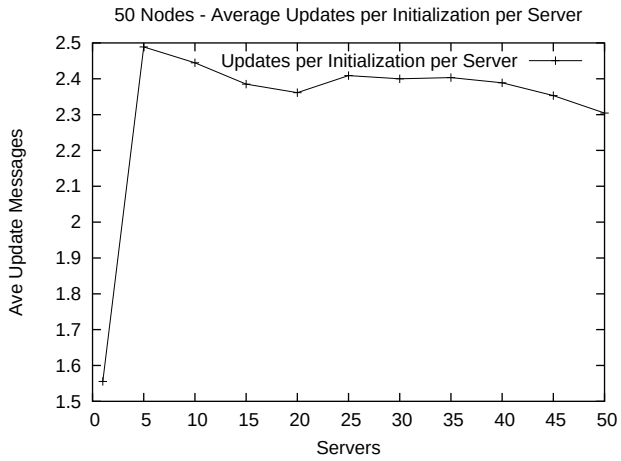

Figure 6.14: Proactive Update traffic per server in a network of 50 nodes as a function of number of servers.

The cost of queries does not decrease as fast with the number of servers. This is shown in Figure 6.15. The reason behind this is again the lower connectivity and the inability of the protocol to reliably build the update grid. This is further shown in Figure 6.16 which shows that the success rate grows slowly with the number of servers because of the increased difficulty of constructing the update grid in the low-density environment and the lower reliability of the queries.

6.2 Effects of Mobility

Mobility is a property of ad hoc networks that leads to problems for many systems deployed in such networks. It is important that mobility does not have a significant effect on the performance of our protocol or its adaptability and efficiency in such an environment will be questionable.

The network under examination is a moderately dense network of 100 nodes with 250 meters transmission range, deployed in an area of 2000 by 2000 meters. Several factors played a role in selecting such a network topology. First, a network of such density is of higher possibility of deployment than the very dense 500 node networks. Also, the *ns-2* network simulator used to run the simulations imposes the use of a less dense network topology since it does not scale well with higher numbers of mobile nodes. To measure the effects of node mobility, scenarios with node speeds of 10 m/s, 20 m/s, and 30 m/s were compared against the data from a static network. The same measures of overhead traffic and query efficiency are used to analyze performance as described

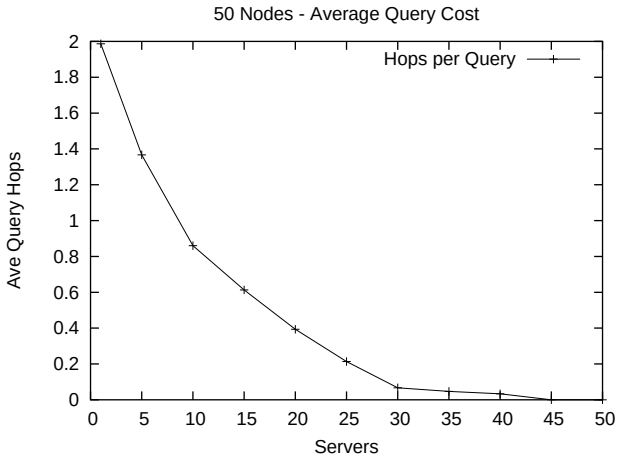

Figure 6.15: Query messages generated per application layer query in a network of 50 nodes.

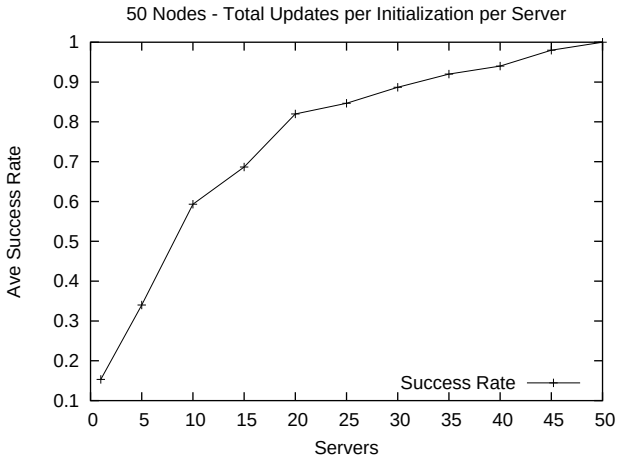

Figure 6.16: Query success rate in a network of 50 nodes as a function of the number of servers available on the network.

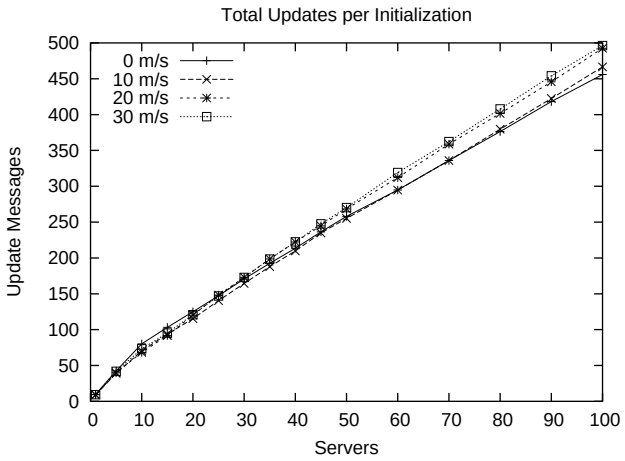

Figure 6.17: Proactive Update traffic in a network of 100 nodes as a function of number of servers for different node speeds.

in Section 6.

Figures 6.17 and 6.18 show that overhead traffic remains virtually unchanged when mobility is introduced into the network. We stipulate that the small fluctuations in the data are due to the different network topologies and not the presence of node mobility. Thus, our protocol performs well under conditions where node mobility exists. Adaptability was one of the requirements of our protocol and it has been met according to our measurements.

The cost of locating content does not seem to vary significantly with the changes in node speed. As Figure 6.19 shows, the hops travelled by query messages per location attempt is similar for the static topology and the three dynamic topologies.

An interesting phenomenon occurs when considering query success rates. As Figure 6.20 shows, increased node mobility actually seems to lead to minimal increases in query success rates for small numbers of servers. This is due to the fact that mobile nodes will lead to location information being present in more parts of the network as nodes formerly located along the update grid travel through the network and answer queries.

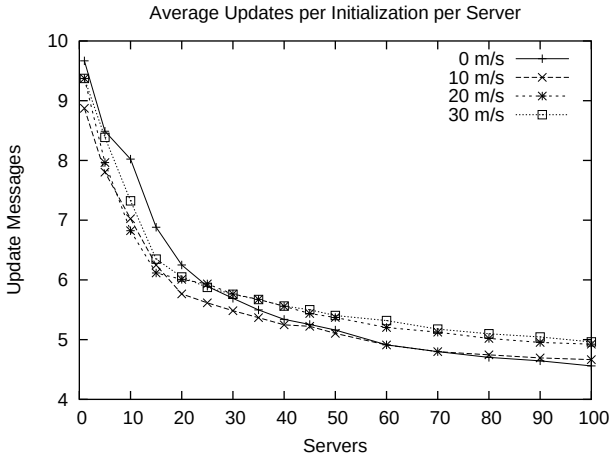

Figure 6.18: Proactive Update traffic per server in a network of 100 nodes as a function of number of servers for different node speeds.

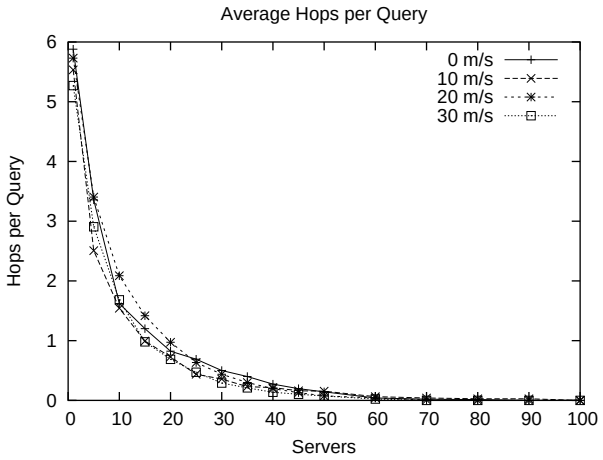

Figure 6.19: Query messages generated per application layer query in a network of 100 nodes for different node speeds.

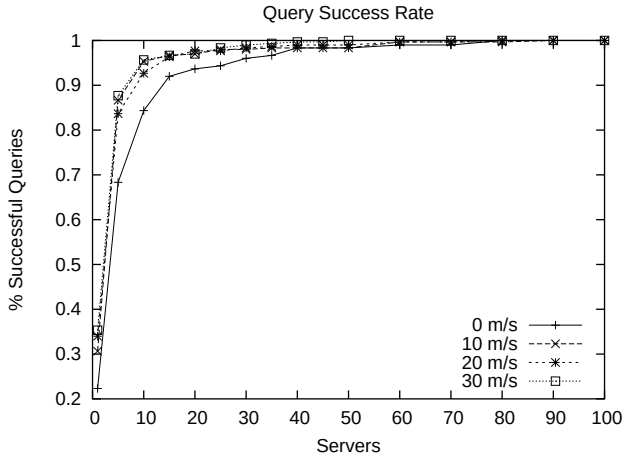

Figure 6.20: Query success rate in a network of 100 nodes as a function of the number of servers available on the network for different node speeds.

Chapter 7

Optimizations

This chapter sets forth a number of possible optimizations to *GCLP*. These optimization are the subject of future work and have not been studied extensively from the point of view of performance in real-world conditions.

7.1 Suppressing Update Initiations

This optimization attempts to remove the minimum number of updates sent by a server. In the protocol described above, each node will send at least one *UM* to each sector given the presence of nodes in that sector. However, this may lead to larger overhead traffic even if there is an abundance of servers in the same sector. To lower such proactive traffic, a node may instead decide not to initiate an update in a given direction if it knows that there are other servers already located in that sector. In this scenario, a node will keep track of which of its neighbors are also servers. It would then not initiate an update to a sector that already hosts a server. However, in this case, care should be taken to ensure that too many servers do not suppress their *UM*'s since this may lead to decreased fault-tolerance due to loss of redundancy.

This would lead to smaller overhead traffic in the network as nodes will use information gathered to adapt their proactive behavior. The resulting overhead traffic will be proportional to the dimensions of the network and not to the number of servers because servers located on the same trajectory will no send duplicate updates, resulting in overhead traffic that only spans each trajectory once. However, this may also have some negative implications. For example, the desired property that hosts will always find servers closest to them may now be lost under some mobility

conditions as servers may decide not to initiate updates because of other servers that have already left their sector.

7.2 Avoiding Empty Spots in Network

As described in Chapter 4, the protocol does not take into account the possibility of empty regions along update or query trajectories. When a node has to select the next hop in such a trajectory and it notices that there are no neighbors in the desired sector, it stops the transmission thus interrupting the trajectory at that point. A possible optimization that aims at rerouting trajectories around empty regions in the network may be employed to increase fault-tolerance. When a node chooses a next hop in a trajectory and notices that it has no neighbors in the desired sector, instead of interrupting the trajectory, it may lookup neighbors in the adjacent sectors and pick one of them to continue the trajectory. This may not be done more than several consecutive times (say, at most 3) because the deviated trajectory may be poor and the empty region may well be the edge of the network.

This modification would increase fault-tolerance in the protocol as more *CLS*'s will exist and empty regions will no longer isolate parts of the network where servers may be present from others that may be poor in servers. It will be especially useful in sparse network where such empty regions are more common.

7.3 Other Rating Algorithms for Selecting Next Hop

Section 4.2.2 describes several heuristics for selecting next hop in trajectories. A number of other algorithms are possible that may give more weight to the distance between the nodes or to the deviation from the perfect trajectory (e.g., rating $R = d^2/r$). Also possible is the inclusion of other factors in the algorithm. For example, the age of entries in the neighbors table may be used to give more weight to more recent entries. This may affect the selection process when node mobility is present.

Chapter 8

Conclusion

In this thesis we have presented a protocol for content discovery in location-aware mobile ad hoc networks. The protocol, Geography-based Content Location Protocol, *GCLP*, uses location information to achieve scalability and cost effectiveness as measured by distance between clients and discovered servers. We have also presented a mathematical analysis of our protocol as well as results from our simulations. Our analysis and simulation results demonstrate that *GCLP* is indeed a viable and efficient content discovery scheme.

References

[1] "www.napster.com."

[2] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*, pp. 149–160, 2001.

[3] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, UC Berkeley, Apr. 2001.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–??, 2001.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pp. 161–172, ACM Press, 2001.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," *Lecture Notes in Computer Science*, vol. 2009, pp. 46–67, 2001.

[7] E. Royer and C. Toh, "A review of current routing protocols for ad-hoc mobile wireless networks," *IEEE Personal Communications*, Apr. 1999.

[8] E. D. Kaplan, ed., *Understanding GPS: Principles and Applications*. Boston, MA: Artech House, 1996.

[9] Y.-B. Ko and N. H. Vaidya, "Location-aided routing(lar) in mobile ad hoc networks," in *IEEE MobiCom 1998*, 1998.

[10] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Mobile Computing and Networking*, pp. 243–254, 2000.

[11] B. Nath and D. Niculescu, "Routing on a curve," in *HOTNETS-I*, (Princeton, NJ), October 2002.

[12] I. Aydin and C.-C. Shen, "Facilitating match-making service in ad hoc and sensor networks using pseudo quorum," in *11th IEEE International Conference on Computer Communications and Netwroks (ICCCN)*, (Miami, FL), October 2002.

[13] H. Chen, D. Chakraborty, L. Xu, and T. Joshi, "Service discovery in the future electronic market," in *Proc. Workshop on Knowledge Based Electronic Markets, AAAI*, 2000.

[14] H. Chen, A. Joshi, and T. W. Finin, "Dynamic service discovery for mobile computing: Intelligent agents meet jini in the aether," *Cluster Computing*, vol. 4, no. 4, pp. 343–354, 2001.

[15] V. V. Sumi Helal, Nitin Desai and C. Lee, "Konark a service discovery and delivery protocol for ad-hoc networks," in *Third IEEE Conference on Wireless Communication Networks (WCNC)*, (New Orleans, LA), March 2003.

[16] D. Doval and D. O'Mahony, "Nom: Resource location and discovery for ad hoc mobile networks," in *Med-hoc-Net 2002*, (Sardegna, Italy), September 2002.

[17] O. Ratsimor and D. Chakraborty, "Allia: Alliance-based service discovery for ad-hoc environments," in *ACM Mobile Commerce Workshop*, September 2002.

[18] E. Guttmann, C. Perkins, J. Veizades, and M. Day, "Service location protocol," *IETF Internet Draft, RFC 2608*, 1999.

[19] S. D. Gribble, M. Welsh, J. R. von Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao, "The ninja architecture for robust internet-scale systems and services," *Computer Networks*, vol. 35, no. 4, pp. 473–497, 2001.

[20] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *Mobile Computing and Networking*, pp. 24–35, 1999.

[21] A. I. T. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Symposium on Operating Systems Principles*, pp. 188–201, 2001.

[22] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, "Distributed object location in a dynamic network," Tech. Rep. UCB/CSD-02-1178, Apr. 2002. Updated version to appear in SPAA 2002.

[23] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proceedings of ACM ASPLOS*, ACM, November 2000.

[24] Y. Chen, R. H. Katz, and J. D. Kubiatowicz, "Scan: A dynamic, scalable, and efficient content distribution network," in *Proceedings of the International Conference on Pervasive Computing (Pervasive 2002)*, (Zurich, Switzerland), August 2002.

[25] L. Cheng and I. Marsic, "Service discovery and invocation for mobile ad hoc networked appliances," in *Proceedings of the 2nd International Workshop on Networked Appliances (IWNA2000)*, (New Brunswick, NJ), November 2000.

[26] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "Gsd: A novel groupbased service discovery protocol for manets," in *4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN 2002)*, 2002.

[27] C. Perkins, "Ad-hoc on-demand distance vector routing," in *MILCOM '97 panel on Ad Hoc Networks*, 1997.

[28] Q. Z. B. L. Jiangchuan Liu, Kazem Sohraby and W. Zhu, "Resource discovery in mobile ad hoc networks," in *Handbook on Ad Hoc Wireless Networks*, CRC Press, 2002.

[29] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *MOBICOM'02*, (Atlanta, GA), September 2000.