Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-002

Enhancing location service scalability with HIGH-GRADE

Yinzhe Yu, Guor-huar Lu, and Zhi-li Zhang

January 12, 2004

Enhancing Location Service Scalability with HIGH-GRADE

Yinzhe Yu
Department of Computer Science and Engineering
University of Minnesota
yyu@cs.umn.edu

Guor-Huar Lu
Department of Electrical and Computer Engineering
University of Minnesota
ghlu@ece.umn.edu

Zhi-Li Zhang
Department of Computer Science and Engineering
University of Minnesota
zhzhang@cs.umn.edu

ABSTRACT

Location-based routing significantly reduces routing overheads in ad hoc networks by utilizing position information of mobile nodes in making forwarding decisions. *Location service* is therefore critical to location-based routing, the scalability of which hinges largely on the location query and update overheads of such service. Although several location service schemes have been proposed, most of them focus only on one or two aspects of the scalability in their performance evaluation, and a comprehensive comparative study is missing. In this paper, we first explore the design space of location service for location-based ad hoc routing and discuss the tradeoffs involved in various design choices. We then propose HIGH-GRADE, a new location service scheme that employs a multilevel hierarchical location server structure and a multi-grained location information organization. We develop a uniform theoretical framework to analyze HIGH-GRADE and four other existing schemes in terms of three metrics: location maintenance cost, location query cost, and storage requirement cost. With both theoretical analysis and simulation experiments, we show that HIGH-GRADE demonstrates superior scalability, especially when a *localized* data traffic pattern is assumed, in which case all the three scalability metrics are bound by $O(v \log N)$.

General Terms

Algorithm, Analysis, Performance

Keywords

Location service, scalability, ad hoc network

1. INTRODUCTION

The future mobile computing environment will consist of numerous heterogeneous devices, many of which are mobile, all able to communicate with each other. As a key enabling technology to such environments, ad hoc network has two important features. First, it does not require a network infrastructure to be built in advance, which can be both slow and expensive. Moreover, ad hoc networks should have the ability to self-configure, thereby requiring little user intervention. To fully realize the potential of ad hoc networks, a critical issue to be addressed is the *routing scalability*. Unlike infrastructure-based wired networks, routing in an ad hoc network is performed by each and every node in a cooperative manner. Due to limited resources such as power, bandwidth, processing capability and storage space at the nodes as well as mobility, reducing routing overheads while ensuring high packet delivery rate is critical in an ad hoc network. Despite many recent advances, design of scalable routing protocols that can effectively operate in an ad hoc networking environment with a large number of mobile nodes remains to be a challenging research problem [1].

Classical ad hoc network routing protocols, such as DSR[2] and AODV[3], focus their design on accommodating the mobility of nodes. In these protocols, nodes attempt to discover routes *on demand* by sending route query messages that flood the entire network. As the network size grows, the cost of flooding becomes prohibitive. Therefore, the performance of these protocols is generally not satisfactory for networks with more than a few hundred nodes [4].

To address the scalability problem associated with the early ad hoc network protocols, location-based routing protocols have been proposed [5, 6, 7]. Location-based routing assumes the availability (e.g., via the GPS system) of geographical location information of nodes: nodes (source or intermediate nodes) make forwarding decisions based on the location of destination, e.g., by choosing the neighbor closest to the destination. Hence nodes in location-based routing do not need maintain the traditional "routing tables" about all potential destinations, only information about neighbors.

While location-based routing eliminates the cost associated with route discovery and maintenance for particular destinations, it introduces a new problem, namely, *the need for a location service*: before packets can be forwarded, a (source) node needs to discover the location of a destination node; location information of nodes must be maintained and updated, as nodes move around. In ad hoc networks, location service is a co-operative "peer-to-peer" service in which network nodes are both users and servers. Each node stores its location information on some other nodes, called its *location servers*. As nodes move around, they need to update the location information on their location servers through *location update* operations. When a node needs the location information of a destination node, it performs a *location query* operation, which uses certain rules to find the appropriate location server(s) of the destination node and retrieves the location information. Hence location-based routing shifts the routing scalability from route discovery and maintenance to location discovery and maintenance, and *scalability of location service is critical to the overall scalability of location-based ad hoc routing.*

Several location service schemes have been proposed in the literature, GLS[7], SLURP[8], SLALoM[9], DLM[10], and Hierarchical Grid[11] are five representative examples. Although scalability of location service is the main design con-

cern in all these schemes, they employ different performance metrics. For instance, [11] focuses mainly on the location update cost (i.e., average number of packets forwarded per second in the network to update all the location servers), while in [9] and [10], both location update and location query costs are considered. In addition, different sets of environmental parameters in (e.g., mobility pattern and traffic pattern) are used in simulation evaluation of these schemes, making direct comparison of their scalability and performance rather difficult. In addition to location query and update overhead, which is a good indicator of the CPU processing power and battery power consumption that a location service requires, we believe that memory/storage requirement of location service is also an important metric that should be taken into account in the design of location service for mobile ad hoc networks, especially when a large number of small mobile devices are involved. Furthermore, scalability of location services should be considered under various traffic patterns (e.g., communications patters among nodes), not solely under uniform traffic patterns. Of particular importance are more *localized* traffic patterns, i.e., nodes are more likely to communicate with those nodes that are close-by than those that are far away. Such traffic patterns have proved to be prevalent among many kinds of human or computer communications environments, which we believe is also likely to hold in ad hoc network environments.

Towards the goal of better understanding and enhancing the scalability of location service in ad hoc networks, in this paper we first explore the design space of location service along several dimensions, and categorize the existing location service schemes along these dimensions. We then propose and develop a new location service scheme, called *HIearchical Geographic Hashing with multi-GRained Address DElegation (HIGH-GRADE)*, that combines the advantages of the existing schemes. We also present a common theoretical framework for studying the scalability of location service schemes, and provide a uniform set of criteria for comparative analysis and evaluation of location services in ad hoc networks. Based on this theoretical framework, we analyze the performance and scalability of HIGH-GRADE as well as GLS, SLURP, SLALoM and DLM. We show that the HIGH-GRADE exhibits superior scalability performance in terms of both location update/query overhead and memory/storage requirement, in particular, under the assumption of localized traffic patterns. The theoretical results are also supported by simulation evaluation. Our study provides valuable insight into the design of scalable location services and sheds light on the comparative performance of various alternatives.

The remainder of the paper is organized as follows. Section 2 discusses the design space and tradeoffs involved in location service design. In section 3, we present the details of HIGH-GRADE. A comprehensive comparison of HIGH-GRADE and four other schemes based on theoretical analysis is described in section 4. Section 5 is a simulation comparison of HIGH-GRADE and GLS[7]. Finally, we conclude the paper in section 6.

2. LOCATION SERVICE: THE PROBLEM AND THE DESIGN SPACE

The basic location service problem can be described as follows:

Table 1: Notation of Section 2

N	number of nodes
A	area of the entire network region
H	total number of hierarchy in network structure
γ	node density
r_t	node transmission radius

> *For a node (with ID) B wishing to communicate with another node (with ID) A, how to discover the current location of A, (i.e., how to map a node's ID to its current location in a distributed and dynamic manner)? More specifically, how should A choose a set of nodes as its location servers? How should A update its location servers to ensure freshness of the information? How should B find the appropriate servers to query for A's location?*

Consider a mobile ad hoc network with N nodes, each with a unique ID, uniformly distributed in a squared region with area of A (i.e., the side length of the network area is $\sqrt{A}$). Our basic design goal is to make the location service scheme scales well as the network size grows, in particular, as N grows. We want the location update/query processing overhead and the memory requirement on individual nodes grows modestly as N grows. To focus our analysis on the scalability in terms of N, we further assume that the network area A grows linearly with N, i.e., the node density γ, defined as $\gamma = \frac{N}{A}$, remains constant as N grows. We also assume that each node has an omni-directional antenna, and the transmission range of all the nodes are set to be the same value r_t. The transmission range is set such that the network is well connected.

We next describe the possible design space of a location service along several dimensions, and classify the previous schemes as well as our HIGH-GRADE scheme accordingly.

2.1 Location server organization structure

The first question in designing a location service is which nodes to select as the location server(s) for a node A. We call this process the location server organization. The simplest organization structure is a *flat structure*, which is used in SLURP[8]. In this method, the network area is divided into a flat grid of "squares". A hash function is applied to node A's ID to obtain the (x, y) coordinates of a point in the entire network area. The square containing that point is defined as A's "home square". A's location information is stored on all the nodes in its home square. To query A's location, a node B applies the same hash function to A's ID to find out A's home square. B can then forward a query packet to that square to retrieve the location of A. Figure 1 illustrates this method, where the solid line represents the location update path and the dashed line represents the location query path.

A well-known drawback of the flat structure is that the expected distance a query packet from B needs to travel to retrieve A's location grows as the network size grows. In particular, even if B is geographically close to A, its query packet may still need to travel a long distance. To address this problem, a *two-level structure* is used in SLALoM and DLM. In these schemes, the entire network area is divided into many level-2 squares. Node A's ID is hashed to a (same) point in each of the level-2 squares. Node A thus has one

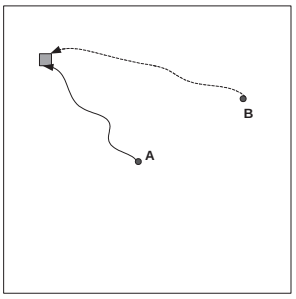

Figure 1: Flat server organization structure

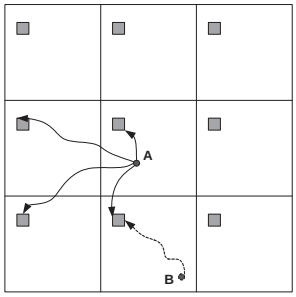

Figure 2: Two-level server organization structure

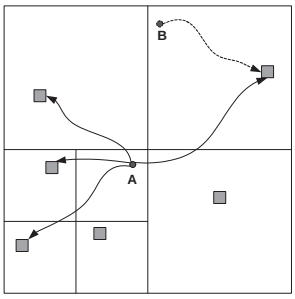

Figure 3: Multilevel hierarchical server organization structure

home square in every level-2 squares. The distance a query packet needs to travel is then bounded by the size of a level-2 square. Figure 2 illustrates this method.

The two-level structure reduces the location query cost, but increases the cost of updating location servers. Instead of updating only one home square as in the flat structure, many home squares need to be updated in the two-level structure. A characteristics of the two-level structure is that the location servers are distributed uniformly across the network. A design alternative is to distribute the servers such that they are denser in the area close to node A and sparser otherwise. In this way, nearby nodes of A incur low cost in querying A's location, while far apart nodes pay a higher cost. Such design is especially attractive for localized traffic pattern, where many queries are from nearby nodes. This method, which we call as *multi-level hierarchical structure*, is adopted by GLS and our HIGH-GRADE scheme. GLS assumes a square shape network area, which we call the level-H square (where H is the total number of hierarchies in the network). The level-H square is divided into four level-(H–1) squares, each of which are further divided into four level-(H–2) squares, and so on, until finally reaching the level-0 squares. In each level-i square (for $i > 0$), node A selects three location servers, one in each level-(i–1) square quadrants that A is not in. The structure of a GLS network is illustrated in Figure 3. The HIGH-GRADE design of network structure is based on the same philosophy, but also has salient differences from GLS, which we will explain in Section 3.

A notable difference between GLS and the other schemes is the way it chooses a location server in that server's service area (e.g., a quadrant in GLS, or a level-2 square in SLALoM). GLS uses the following method to choose a node A's server: the chosen server C has the smallest ID that is larger than A's ID, i.e.,

$$C = \min\{x | \text{ node } x \text{ is in the quadrant}, ID(x) > ID(A)\},$$

(C is called a "closest" node of A in the quadrant). On the other hand, all the other schemes choose servers by geographical closeness to a point obtained using hash functions.

2.2 Granularity of location information

Once a node A decides which nodes to serve as its location servers, A will update these servers periodically. How frequently a server needs to be updated depends on the *granularity of location information* stored on that server. The straightforward method is to store the exact location on all the servers, which we call the *single grained* strategy. This strategy is adopted by SLURP and GLS. The drawback of this strategy is that the information about the exact location of a node becomes stale quickly, requiring frequent updates, and frequent updates to remote servers are expensive. Trying to mitigate this problem, GLS has an option to update remote servers infrequently. In order to compensate the stale information stored on remote servers, node A leaves a "pointer" in every level-0 squares it traverses (to indicate its new location) between location update periods. However, as the network size grows larger, the number of pointers each node left grows accordingly, and this option may create more problem than it solves. In this paper, we will consider the GLS scheme in which all the servers are updated with the same period.

SLALoM employs a *two-level grained* location information. To do this, SLALoM defines "home squares *near* A" as the nine home squares closest to A, i.e., the home square in the level-2 square where A is in, and the home squares in the eight immediate surrounding level-2 squares. In SLALoM, all the home squares of A ("near A" or not) knows which level-2 square A is in (the coarse grained location). In addition, all the home squares *near* A knows the exact location of A (the fine grained location). Therefore, as A moves around, nearby servers need to be updated relatively frequently, while remote servers need only relatively infrequent updates.

To carry the idea of location granularity a step further, HIGH-GRADE adopts a strategy of using *multi-grained* location information. In HIGH-GRADE, a node A has location servers at each level-i square it resides in. Instead of storing the exact location of A on all these servers, a rough information of "which level-($i-1$) square A is in" is stored on a server serving the level-i square. In this way, location servers at higher level squares (i.e., those far away from A) need only infrequent updates. On the other hand, by tracing a series of servers at lower and lower levels, the exact location of A can be obtained eventually.

Finally, the DLM scheme offers two options in location granularity. Either the "complete address" or a "partial address" can be stored at a server. The partial address method is similar to HIGH-GRADE's multi-grained location information.

2.3 Design tradeoffs

In Figure 4, we summarize the taxonomy of the five location service schemes over two dimensions of design space—server

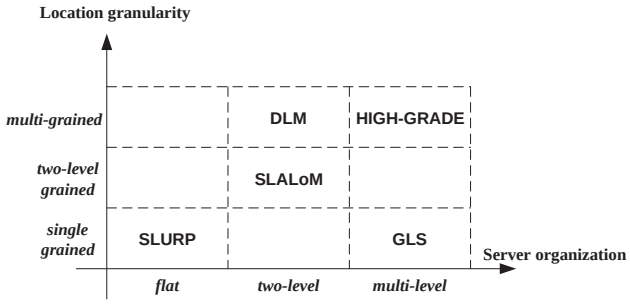

Figure 4: Taxonomy of location service schemes along two dimensions

organization structure and location information granularity. As we have seen, many tradeoffs are involved in making these design choices. For example, changing the server organization from flat to two or multiple levels can reduce the cost of location query. However, since it duplicates more copies of location information for each node, the average amount of location information each server has to store increases accordingly, thereby increasing the storage overhead of each node. Storing coarse grained location information on location servers reduces the location update cost. However, with coarse grained locations, multiple servers need to be queried to assemble the exact location of a node. It therefore increases the number of steps involved in the location query process.

3. THE HIGH-GRADE SCHEME

In our design of the HIGH-GRADE scheme, a basic premise is that data traffic in large networks exhibits a pattern of *locality*, i.e., traffic between nearby nodes are much heavier than that between far away nodes. We therefore place the location servers of a node A in a "multi-level hierarchical" structure, such that they are dense in the nearby area of A and sparse in remote area. In addition, to reduce the location update cost, servers at different levels (distances from A) store A's location information in different granule levels. As we shall see, the multi-grained location information reduces the location update cost dramatically, while increases the location query cost by a constant factor only.

We now first introduce a technique called *geographically scoped consistent hashing (GSCH)*, which is a multi-level hierarchical server organization method used by HIGH-GRADE to determine location servers. We then describe the location update and query process in HIGH-GRADE. Finally, we present some techniques to enhance the performance of the basic HIGH-GRADE scheme.

3.1 Geographically scoped consistent hashing

As discussed in Section 2, we consider a mobile ad hoc network with N nodes in a square region with area A. In HIGH-GRADE, we recursively divide the network area into a quad-tree-like hierarchy of squares. At the top level, the entire area is divided into four quadrants, each of which is further divided into four quadrants as well, and so forth. We denote the level of hierarchy by H. At the bottom level, we have 4^H number of level-0 squares. If we denote the length of the side of a level-0 square as R, then we have $R = \frac{\sqrt{A}}{2^H}$. Figure 5 illustrates this hierarchy of squares with an example

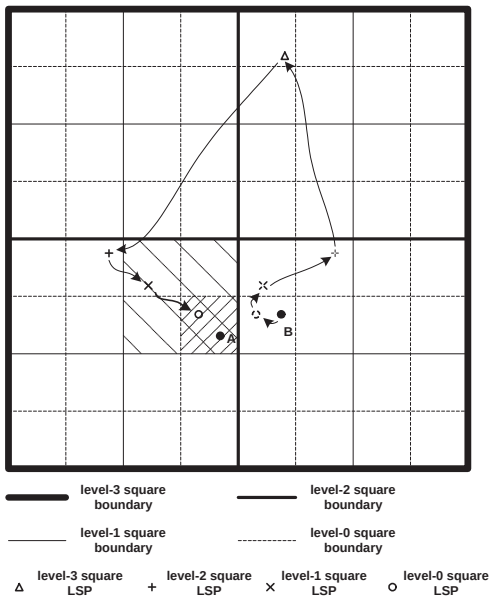

Figure 5: Network hierarchy and location query in HIGH-GRADE

where $H = 3$.

We note that in such a network hierarchy, *each node A resides in exactly one level-i square for $0 \leq i \leq H$* . If we apply a Cartesian coordinates system to our network area, with the lower left point as the origin, then we can formally define the "level-i square of a node A" as follows.

DEFINITION 1 (LEVEL-i SQUARE). *Let the current coordinates of a node A be (x_A, y_A). The level-i square of A is defined as a square area with lower left point (called its origin) $(x^o_{A,i}, y^o_{A,i})$ and side length $2^i R$, where*

$$x^o_{A,i} = x_A - (x_A \bmod (2^i R)),$$
$$y^o_{A,i} = y_A - (y_A \bmod (2^i R)).$$

Definition 1 means that when we consider a point A in a level-i square, the absolute coordinates (x_A, y_A) can be decomposed into two parts, the absolute coordinates of the level-i square *origin* and the *relative position* of the point in the square. Figure 5 illustrates the level-0 and level-1 squares of node A with shadowed lines.

To distribute location servers such that they are more and more sparse as one moves away from node A, A selects one location server (set) at each level-i square in HIGH-GRADE. To explain how servers are chosen in a level-i square, we first define the concept of a *location server point (LSP)*.

DEFINITION 2 (LOCATION SERVER POINT). *A node A's location server point in its level-i square is defined as the point whose coordinates are $LSP_{A,i} = \langle x^l_{A,i}, y^l_{A,i} \rangle = \langle x^o_{A,i}, y^o_{A,i} \rangle + h_i(ID(A))$, where $\langle x^o_{A,i}, y^o_{A,i} \rangle$ is the origin of A's level-i square, and h_i is a uniform hash function that hashes a node's ID to a relative position in a level-i square.*

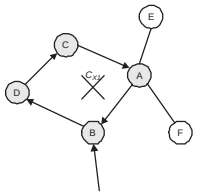

Figure 6: Perimeter based server selection

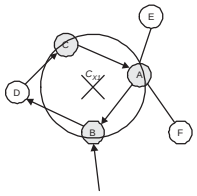

Figure 7: Perimeter and distance based server selection

In Definition 2, we assume that there are $H+1$ *well-known* uniform hash functions $h_0, \ldots, h_H$; therefore, they can be used by all the nodes in the network. In addition, since $H+1$ different hash functions are used, the LSP's of a node A will have different relative positions in different level of squares. In Figure 5, node A's four LSP's are identified by four different markers (with solid drawings). As we can see, each LSP is a hash point within a geographically scoped area, i.e., a level-i square. We therefore call this technique *geographically scoped hashing*. To achieve balanced load on location servers and robustness over node failures, we design the server selection method around LSP's with the same philosophy of "consistent hashing"[12], as described in the next subsection. Many other good properties of consistent hashing can be found in [13]. Combining these two methods, we call this technique *geographically scoped consistent hashing*.

3.1.1 Server Selection around LSP

A LSP provides a rendezvous point for a pair of node A and B at which B tries to find A's location. However, typically there may not exist a node at the exact position of the LSP. Therefore, we need a *consistent* way of choosing location servers around the LSP, so that location updates and location queries result in the same node(s). Fortunately, this is relatively easy to achieve.

In GPSR [5], packets can operate in two modes: greedy mode and perimeter mode. When an intermediate node has a neighbor that is closer to the destination, it adopts the greedy strategy and forwards the packet to the neighbor closest to the destination. Otherwise, the packet is set to the perimeter mode, and is forwarded according to the "right hand rule" based on a planar subgraph of the network topology. Assuming GPSR as our underlying forwarding mechanism, in our case when there is no node at the exact position of the LSP, a packet targeting the LSP will circulate on the perimeter around the LSP, as shown in Figure 6. We can use the following two strategies to select location servers around the LSP's.

1. Store the location information on all the nodes on the perimeter enclosing the LSP. This strategy is used in [14] for sensor networks. It is illustrated in Figure 6.
2. Store the location information on those nodes that are on the perimeter and within a distance threshold κ of the LSP. This strategy is illustrated in Figure 7. A good property of this strategy is that it creates an upper bound on the average number of nodes that are selected around one LSP when the node density γ is constant.

3.2 Multi-grained location information

The HIGH-GRADE scheme stores multi-grained location information on servers at different levels (i.e., servers around LSP's in different level of squares). Specifically, for $1 \leq i \leq H$, at a level-i server of node A, we store the information about the level-$(i-1)$ square that A resides in(one of the four level-$(i-1)$ squares in the current level-i square); we store A's exact location only on its level-0 servers. The following definition gives the exact format of the location information records stored on the servers.

DEFINITION 3 (LOCATION RECORD). *Suppose node C is a level-i server of node A, then the location information record stored on C, denoted $Record(C, A)$ has the following format:*

$$Record(C,A) = \begin{cases} (A, (x^o_{A,i-1}, y^o_{A,i-1}), i-1) & \text{if } 1 \leq i \leq H; \\ (A, (x_A, y_A), -1). & \text{if } i = 0. \end{cases}$$

Definition 3 shows that the record stored in non-level-0 servers include the origin of the next lower level square, as well as the level number. These two fields uniquely specifies the level-$(i-1)$ square node A is in. On the other hand, a record stored in a level-0 server contains the exact location of node A and a NULL value (-1) to indicate that the exact location instead of a square is stored. We note that with this design, once a node B finds one location server of node A at some level, it can trace a sequence of servers down the hierarchy at lower and lower level to find the exact location of A.

3.2.1 location update and maintenance

From Definition 3, we can see that as long as node A remains in its level-i square, all its level-j servers for $j > i$ have the current information, and need not be updated. This gives the following rule for location updates in HIGH-GRADE.

RULE 1 (LOCATION UPDATE). *A level-j server of node A needs to be updated when and only when A moves across a level-i square boundary for $i \geq j-1$.*

Note that nodes in an ad hoc networks are mobile. After a set of servers around a LSP are updated with the current location information, some servers may move away from the LSP, while other nodes may move close and become a perimeter node. Therefore, we need a mechanism to maintain the location information on the set of servers around the LSP. To accomplish this task, we use the perimeter refresh protocol as described in [14]. In that protocol, one server on the perimeter is elected as the "home node" of the LSP (e.g., the first node on the perimeter when the perimeter is traversed is elected). The home node will refresh the

perimeter periodically by sending update messages towards the LSP point. When the home node moves away so that it is no longer on the perimeter, a new home node is elected on perimeter refreshing.

3.2.2 Location Query

We now describe how a node B finds the location of A through A's location servers. We first define the concept of *minimal common square (MCS)* of two nodes.

DEFINITION 4 (MINIMAL COMMON SQUARE). *As defined in Definition 1, for $0 \leq i \leq H$, node A's level-i square has the origin $\langle x^o_{A,i}, y^o_{A,i}\rangle$; node B's level-i square has the origin $\langle x^o_{B,i}, y^o_{B,i}\rangle$. The minimal common square of A and B is defined as A's level-j square such that*

$$j = \min\{k | x^o_{A,k} = x^o_{B,k}, y^o_{A,k} = y^o_{B,k}\}.$$

Intuitively, as we examine A and B's level-i squares from small i value to large i value, the MCS of A and B is their first coincide level-i square (i.e., the first common level-i square). Note that the MCS of a pair of nodes always exists since $x^o_{A,H} = x^o_{B,H} = 0$ and $y^o_{A,H} = y^o_{B,H} = 0$, i.e., the level-$H$ square is always a common square. The MCS of A and B is important because if B knows their MCS is at level-i, then B can obtain $LSP_{A,i}$ immediately by adding $h_i(ID(A))$ to the origin of the MCS.

Unfortunately, B has no way to know *a priori* the MCS. Therefore, it may need to make several unsuccessful attempts before it finds the first LSP of A. To explain this process, we first define the concept of *potential location server point (pLSP)*.

DEFINITION 5 (POTENTIAL LSP). *A pLSP obtained by node B for querying A's location at a level-i square is defined as:*

$$pLSP_{B,A,i} = \langle x^l_{B,A,i}, y^l_{B,A,i}\rangle = \langle x^o_{B,i}, y^o_{B,i}\rangle + h_i(ID(A))$$

where $\langle x^o_{B,i}, y^o_{B,i}\rangle$ is the origin of B's level-i square.

Note that the same set of hash functions h_i's are used in both LSP and pLSP calculation. In LSP, the hashed point is applied to A's level-i square, while in pLSP, B's level-i square is used. Obviously, if the level-i squares of A and B are the same, then $LSP_{A,i} = pLSP_{B,A,i}$. By the definition of LSP, pLSP, and MCS, we have the following property:

If the MCS of A and B is a level-i square, then

$$i = \min\{k | LSP_{A,k} = pLSP_{B,A,k}\}$$

This property suggests one strategy for B to find A's location: to query $pLSP_{B,A,k}$ sequentially for increasing k. Suppose the MCS of A and B is a level-i square, then the first i queries will fail as $pLSP_{B,A,k} \neq LSP_{A,k}$ for $k < i$. When a query at $pLSP_{B,A,k}$ fails, the home node at that pLSP re-forwards the query to the next pLSP, i.e., $pLSP_{B,A,k+1}$, until a location server is finally reached (at $LSP_{A,i}$). The location server at $LSP_{A,i}$ can then re-forward the query sequentially to $LSP_{A,k}$ for decreasing value of k, until a server at $LSP_{A,0}$ receives the query and replies B with the current location of A.

Algorithm 1 : n.**recv**(p)

```
1: if n ≠ homenode(p.LSC) then
2:    GPSR(p)
3:    return
4: rec ⇐ lookup(p.target)
5: if rec == NULL then
6:    re-forward p to the next high level pLSC
7: else
8:    if rec.level == −1 then
9:       reply the orginal querying node with exact location
10:   else
11:      re-forward p to the next low level LSC
12: return
```

Algorithm 1 describes how a node n handles a query packet p. First, n checks whether it is the home node for the current packet, if not, n just forwards the packet with normal GPSR algorithm (line 1 to 3). On the other hand, if n is the home node, it looks up the local location database for pacekt p's query target node. If such a record is not found, n re-forwards p to the next level $pLSC$ (line 6); otherwise, depending on whether n is storing the exact location or a square information of the target node, n will reply with the exact location or re-forward p to the next level LSC (line 8 to 11).

Figure 5 shows an example of how the query message is relayed when node B queries the location of node A. Note that in the figure, markers with dashed drawing are pLSP's, and markers with solid drawing are LSP's.

Although it seems that many re-forwarding steps are involved when B queries the location of A in their MCS, we can show the following *worse case upper bound* on the total length a query packet travels.

PROPOSITION 1 (WORST CASE QUERY DISTANCE). *If node A and B's MCS is a level-i square, then in the worst case, the distance traveled by a query packet from B to reach A's level-0 server is bounded by a constant factor to the side length of the MCS.*

PROOF. The query packet is (re-)forwarded sequentially to $pLSP_{B,A,0}$, $pLSP_{B,A,1}$,..., $pLSP_{B,A,i}$(i.e., $LSP_{A,i}$), and then $LSP_{A,i-1}$,...,$LSP_{A,0}$. In each step, the source and destination of the re-forwarding are within a level-j square (j increases from 0 to i and then decreases to 1). Therefore, the worse case distance of each step is $\sqrt{2}$ times the square side length. Denote the distance between two points in a level-j square by d_j, and the total distance as D, we have

$$\begin{aligned} D &= \sum_{j=0}^{i} d_j + \sum_{j=i}^{1} d_j \\ &< 2 \cdot \sum_{j=0}^{i} \sqrt{2} \cdot R \cdot 2^j \\ &< 4\sqrt{2} \cdot R \cdot 2^i \end{aligned}$$

Therefore, the worst case total distance is bounded by $4\sqrt{2}$ times the MCS side length. □

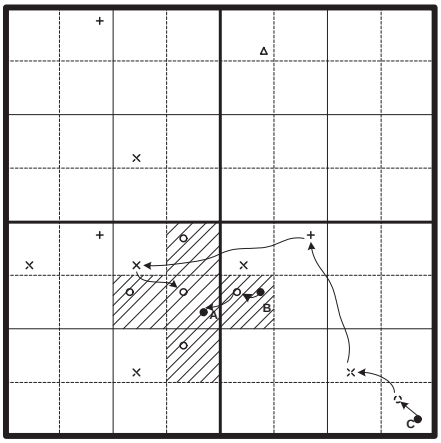

Figure 8: HIGH-GRADE with replicated LSP's

3.3 LSP Replication

As we have discussed, our HIGH-GRADE scheme aims to reduce the location query cost for nodes that are close to each other. However, since a hierarchical square structure is adopted, a pair of close nodes may be separated by a high-level square boundary. For instance, in Figure 5, node A and B are close to each other, but separated by a level-2 square boundary. This results in excessive re-forwarding steps when B queries the location of node A.

To mitigate this problem, we can replicate LSP's at appropriate places to reduce the query steps. Essentially, we trade the overhead in location update cost for the saving in location query cost. Specifically, we use the following rule for LSP's replication.

RULE 2 (LSP REPLICATION). *We replicate each level-i LSP's ($0 \leq i \leq H-1$) of a node A in four of the adjacent level-i squares.*

Figure 8 illustrates how the LSP replication is used. It assumes the same scenario as used in Figure 5, except that A's LSP's are replicated according to Rule 2. The location records stored by the level-0 servers are replicated to the pLSP's in the four surrounding level-0 squares (shaded in the figure). The same is done for the level-1 and level-2 LSP's. Note that in this case, only two steps are involved when B queries location of A, five steps less than in Figure 5. It takes another node, node C, five steps to query A's location, while it would have taken seven if LSP replications are not used.

4. COMPARATIVE STUDY BASED ON ANALYTIC MODELS

In this section, we employ a common theoretical framework to develop analytic models to compare the scalability of HIGH-GRADE with several existing location service schemes, namely, GLS, DLM, SLURP and SLALoM. In particular, we focus on how well these schemes scale as two parameters of the network grows: the *size* of the network, represented by the number of nodes N, and the moving *speed* of nodes, denoted by v. To evaluate the various schemes, we first define three metrics we will use: location maintenance cost, location query cost, and storage requirements cost.

In all the schemes, each node A needs to update its location servers every time a certain event occurs (e.g., crossing square boundary, or moving more than a distance). In addition, in some of the schemes, certain other nodes need to initiate maintenance packets to maintain the location information of A on the servers (e.g., perimeter refreshing in HIGH-GRADE). Both of these packets are used to maintain fresh location information on location servers, we therefore define the *location maintenance cost* to include them both.

DEFINITION 6 (LOCATION MAINTENANCE COST). *The location maintenance cost C_m is defined as the number of forwarding operations each node needs to perform in a second to handle the location update/maintenance packets. It can be viewed as the cost of maintaining fresh location information on location servers in the network.*

Note that in this definition, we measure the location maintenance cost in terms of the forwarding load the location maintenance process puts on each individual node. It is natural to correlate the number of packet forwarding a node performs to the node's CPU processing power and battery power consumption, two important scalability constraints. We emphasize that in the definition, we measure the number of different forwarding operations (hops), rather than the number of different packets. In this way, a location update packet traveling a long distance has a higher cost since it requires more forwarding hops.

Similarly, we define the location query cost as:

DEFINITION 7 (LOCATION QUERY COST). *The location query cost C_q is defined as the number of packet* forwarding operations *due to location queries each node needs to perform in a second.*

We separate the location maintenance and query costs for two reasons. First, as we shall see, the design choices of various schemes usually involve tradeoffs between these two types of cost. Examining these two types of cost separately allows us to derive a better understanding on the consequences of the various design choices. Second, we believe that in a location service scheme, the location query cost is relatively easy to reduce by employing various caching strategies, while the location maintenance cost is not. Therefore, separating the two types of cost provides more information for one to predict the likely scenario in practice.

The third metric we define is the storage requirement cost.

DEFINITION 8 (STORAGE REQUIREMENT COST). *The storage requirement cost C_s of a location service is defined as the number of location records a node needs to store as a location server.*

We point out that all the three metrics are defined in terms of the cost at an *individual* node. Since nodes are all symmetric in the schemes we examine, the expected value of the

Table 2: Notations (cont.)

C_m	location maintenance cost
C_q	location query cost
C_s	storage requirement cost
κ	distance threshold in perimeter refresh
v	node speed
z	average progress of each forwarding hop
ρ_i	level-i square boundary crossing rate
d^u	distance traveled by an update packet
d^q	distance traveled by a query packet
n^u	number of forwarding hops of an update packet
n^q	number of forwarding hops of a query packet
λ	perimeter refreshing rate
P_i^u	prob. querying nodes in level-i square (uniform traffic)
P_i^l	prob. querying nodes in level-i square (localized traffic)
c_1	constant of random distance within a square
c_2	constant of random distance between squares
c_3	another constant of random distance between squares

metrics are the same for all the nodes in the network. We derive these expected values in the remaining part of the section. Before we proceed, we summarize some of the basic assumptions we made:

- As N grows, we assume the geographical area of the network A grows *linearly* with N. This implies that the node density γ of the network is held constant.
- We assume that nodes are moving according to a simplified Random Way-point model [15]. In this model, each node picks a random point in the network area, and moves towards that target point with velocity v. After reaching the target, a new point is selected, and the node continues on.
- A data traffic pattern is the probability distribution of traffic intensities between any pair of nodes in the network. The most commonly used data traffic pattern in evaluating ad hoc networks is the uniform random traffic model. In this model, the probability of initiating a packet between any pair of nodes in the network is the same. Therefore, any time a node has a packet to send, it picks a destination randomly from all other nodes in the network with equal probability. However, it has been shown in [16] that when the uniform traffic pattern is assumed, the end-to-end throughput available to each individual node has a theoretical upper bound of $O(\frac{1}{\sqrt{N}})$. This means that the throughput of the network scales poorly assuming uniform traffic model (no matter what kind of routing scheme is used)! Moreover, the uniform random traffic model does not correlate well to our experience in large real networks, as we commonly observe that local traffic is more intensive than long distance traffic. Therefore, in addition to the uniform traffic model we consider in this section a more localized traffic pattern where data traffic to the nearby nodes are much heavier than to the remote nodes.

The notations introduced in Section 3 and 4 are summarized in Table 2.

4.1 HIGH-GRADE

We derive the expected values of the three metrics for the HIGH-GRADE scheme as functions of N and v.

4.1.1 Location maintenance cost

The location maintenance cost C_m of HIGH-GRADE consists of two parts: the cost due to the location update packets (denoted by C_{m1}) and the cost due to the perimeter refreshing protocol (denoted by C_{m2}). By Rule 1, when a node A moves across a level-i square boundary, an update packet needs to be sent to each of the level-j LSP's for $j \leq i+1$. The following two Lemmas give the frequency of boundary crossing events and the expected number of hops of each location update packet.

LEMMA 2 (BOUNDARY CROSSING RATE). *The level-i square boundary crossing rate (unit $1/s$) of a node A is*

$$\rho_i \approx \frac{\pi v}{2R} \cdot \frac{1}{2^i}, \text{ for } 0 \leq i \leq H-1,$$

where v is the moving speed of the node A, and R is the side length of a level-0 square.

PROOF. First consider the level-0 square boundary crossing rate ρ_0. In [8], the author showed that $\rho_0 \approx \frac{\pi v}{2R}$ by approximating ρ_0 with the crossing rate of a node in a circle area with diameter R. Next, observe that a boundary crossing is either a vertical or horizontal boundary crossing. In either case, a level-i boundary is always also a level-(i–1) boundary, while every other (vertical/horizontal) level-$(i-1)$ boundary is a level-i boundary. Therefore, we have $\rho_i = \frac{1}{2}\rho_{i-1}$ for $1 \leq i \leq H-1$. □

LEMMA 3. *The expected number of hops of a location update packet sent from A to A's level-i LSP is*

$$E(n_i^u) = \frac{c_1 \cdot 2^i R}{z},$$

where c_1 is a constant and $c_1 \approx 0.5214$, and z is a constant representing the average progress towards the destination in each packet forwarding hop.

PROOF. Consider the distance between A and its level-i LSP, denoted by d_i^u ("u" for update). Since we use a uniform hash function h_i to hash A's ID into the LSP, d_i^u can be viewed as the random distance between two points in the level-i square, as shown in Figure 9 (left pane). Therefore,

$$\begin{aligned} E(d_i^u) &= 2^i R \int_0^1\int_0^1\int_0^1\int_0^1 \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}\, dx_1 dy_1 dx_2 dy_2 \\ &= c_1 \cdot 2^i R \approx 0.5214 \cdot 2^i R \end{aligned}$$

A detailed procedures for the integration step above can be found in [17].

The expected number of hops in forwarding the packet is the expected distance divided by z, the average progress of each hop. As shown in [8], z is a function of the radio transmission range r_t and the node density γ. Since we assume both as constants in our model, so is z. Therefore, $E(n_i^u) = \frac{E(d_i^u)}{z} = \frac{c_1 \cdot 2^i R}{z}$. □

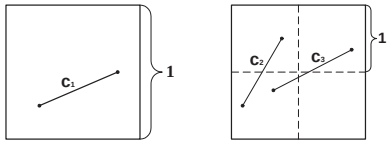

Figure 9: Three constants. Left: random distance between a pair of nodes in a unit square (c_1). Right: random distances between pairs of nodes in two squares (c_2 and c_3).

With Lemma 2 and 3, we are now ready to prove the upper bound of the expected location maintenance cost.

THEOREM 4. *In the HIGH-GRADE scheme, $E(C_m) = O(v \cdot \log N)$, that is, the location maintenance cost metric scales linearly with the node speed and logarithmically with the number of nodes in the network.*

PROOF. As we described above, C_m has two components: C_{m1} and C_{m2}. C_{m1} includes the cost of updating all the $H+1$ LSP's. Therefore,

$$\begin{aligned} E(C_{m1}) &= \sum_{i=0}^{H} \rho_i \cdot E(n_i^u) \\ &= \frac{\pi v c_1 H}{2z} \propto v \cdot H \end{aligned}$$

If we hold R as a constant, then H is proportional to $\log \sqrt{A}$. As $A \propto N$, we have $H \propto \log N$. So $E(C_{m1}) = O(v \cdot \log N)$.

Now consider C_{m2}, the cost of perimeter refreshing. Let the refreshing rate be λ. Assuming the perimeter and distance based perimeter refreshing protocol, the number of nodes around the perimeter is bounded by $\frac{\pi\kappa^2}{\gamma}$, where κ is the distance threshold. Since $H+1$ LSP's need the be refreshed, we have $E(C_{m2}) = (H+1) \cdot \lambda \cdot \frac{\pi\kappa^2}{\gamma} = O(\log N)$. Combining the two components, we have

$$E(C_m) = E(C_{m1}) + E(C_{m2}) = O(v \log N).$$

□

4.1.2 Location query cost

We consider the location query cost metric under the basic HIGH-GRADE scheme, i.e., no LSP replication is used. Our first result is the expected forwarding hops traveled by a location query packet in a level-i square, given in the Lemma below.

LEMMA 5. *If A and B's MCS is a level-i square, then the expected forwarding hops traveled by a location query packet initiated by node B for A's location, denoted by $E(n_i^q)$, has the following formula.*

$$E(n_i^q) = (2^{i+2} - 3) \cdot c_1 \cdot \frac{R}{z}$$

PROOF. As we described in Section 3, the location query packet is re-forwarded sequentially to $pLSP_0$, $pLSP_1$,..., up to $pLSP_i$ (i.e., LSP_i), and then to LSP_{i-1},..., and finally to LSP_0. Note that each of these steps can be viewed as the distance between a pair of random points in a level-j square, where j first increases from 0 to i and then decreases to 1. Therefore, the distance of each step is exactly what we have derived for d_j^u in Lemma 3. We then have,

$$\begin{aligned} E(n_i^q) &= \sum_{j=0}^{i} \frac{E(d_j^u)}{z} + \sum_{j=i}^{1} \frac{E(d_j^u)}{z} \\ &= 2\sum_{j=0}^{i} \frac{c_1 \cdot 2^j R}{z} - \frac{c_1 \cdot R}{z} \\ &= (2^{i+2} - 3) \cdot c_1 \cdot \frac{R}{z}. \end{aligned}$$

□

Lemma 5 gives the expected cost of a location query packet *when A and B's MCS is a level-i square*. To obtain the general expected cost, we need the probability that "the MCS is a level-i square", which we denote by P_i. If the traffic pattern is uniform across the network area, i.e., given A's position, the probability density of B's position is uniform across the network, then we can easily derive that P_i is as follows (we use the notation P_i^u to indicating the uniform pattern).

$$P_i^u = \begin{cases} \frac{3}{4^{H-i}} & \text{if } 1 \le i \le H \\ \frac{1}{4^H} & \text{if } i = 0. \end{cases}$$

To catch the localization property in the traffic pattern, we also define another set of P_i (denoted by P_i^l for "localized" pattern) that is exponentially decaying with i. This means as we move further and further from A, the traffic intensity becomes less and less. We define $P_i^l = \frac{1}{2} P_{i-1}^l$, for $1 \le i \le H$. Given that $\sum_0^H P_i^l = 1$, we have

$$P_i^l = \frac{1}{2^{i+1}} \cdot \frac{1}{1 - \frac{1}{2^{H-1}}}, \text{ for } 0 \le i \le H.$$

With the definition of the two traffic patterns, we can now prove the upper bound of the expect location query cost metric for HIGH-GRADE scheme.

THEOREM 6. *The upper bound of the expected location query cost metric $E(C_q)$ is $O(\sqrt{N})$ for uniform traffic pattern and $O(\log N)$ for localized traffic pattern.*

PROOF. For uniform traffic pattern, we have

$$\begin{aligned} E(C_q) &= \sum_{i=0}^{H} E(n_i^q) \cdot P_i^u \\ &\approx \sum_{i=0}^{H} (2^{i+2} - 3) \cdot c_1 \cdot \frac{R}{z} \cdot \frac{3}{4^{H-i}} \\ &\approx 2^{H+1} \cdot c_1 \cdot \frac{R}{z} \\ &= O(\sqrt{N}). \end{aligned}$$

For the localized traffic pattern,

$$\begin{aligned} E(C_q) &= \sum_{i=0}^{H} E(n_i^q) \cdot P_i^l \\ &\approx \sum_{i=0}^{H} (2^{i+2}-3) \cdot c_1 \cdot \frac{R}{z} \cdot \frac{1}{2^{i+1}} \cdot \frac{1}{1-\frac{1}{2^{H-1}}} \\ &\approx 2 \cdot c_1 \cdot \frac{R}{z} \cdot H \\ &= O(\log N). \end{aligned}$$

□

4.1.3 Storage requirement cost

We now analyze the HIGH-GRADE scheme in terms of the third metric: the storage requirement cost C_s. Remember that C_s is defined as the number of location records a node needs to store as a location server. The following theorem gives the expected value of C_s for the HIGH-GRADE scheme.

THEOREM 7. *The expected value of the storage requirement cost of HIGH-GRADE scheme is:*

$$E(C_s) = O(\log N).$$

PROOF. The average number of records a node stores is the total number of records stored in the network divided by the total number of nodes. Each node stores its location information at $H+1$ LSP's. Assuming the perimeter and distance based server selection method is used, the average number of servers at each LSP is bound by $\frac{\pi\kappa^2}{\gamma}$. Therefore, we have

$$\begin{aligned} E(C_s) &= \frac{N \cdot (H+1) \cdot \frac{\pi\kappa^2}{\gamma}}{N} \\ &= O(\log N) \end{aligned}$$

□

4.2 Analysis of Other Schemes

In this subsection, we analyze the other four schemes, GLS, DLM, SLURP and SLALoM, using the same three metrics under the same set of assumptions.

4.2.1 GLS

As we described in Section 2, the GLS scheme uses a similar multilevel hierarchical structure as HIGH-GRADE. A node A selects 3 location servers in each level-i square, one in each level-$(i-1)$ squares quadrants that the A is not in, as shown in Figure 3. An important difference between GLS and HIGH-GRADE is that GLS stores exact location information on every servers. Therefore, to ensure freshness of location information and reduce the query failure rate, all the location servers need to be updated periodically. The update period is set as the time a node moves a distance of δ.

When a node B wants to find the location of A, it sends a query packet towards a node C_1, the node closest to A in the ID space that B knows of. C_1 does the same, re-sending the query packet to C_2, the node closest to A that C_1 have a record with, so on and so forth, until a location server of A is found. Assuming nodes are relatively static during the lifetime of a packet, the GLS scheme guarantees that in i steps of re-sending, the location server will be reached, where i is the level of A and B's MCS. In addition, in each of the i steps, the source and destination of the re-sent packet in that step are within a level-j square, where j decrease gradually from i to 1.

We now prove the following theorem about the GLS scheme.

THEOREM 8. *For the GLS scheme,*

$$\begin{aligned} E(C_m) &= O(v\sqrt{N}); \\ E(C_q) &= \begin{cases} O(\sqrt{N}) & \textit{for uniform traffic pattern} \\ O(\log N) & \textit{for localized traffic pattern} \end{cases}; \\ E(C_s) &= O(\log N). \end{aligned}$$

PROOF. We first consider the location maintenance cost metric C_m. Since in GLS, a node updates all its servers with the same period, no location maintenance packet (as in the perimeter refreshing of HIGH-GRADE) is needed. C_m is solely due to location updates in the GLS scheme. Consider the expected distances the three update packets traveled to update the three locations servers at the level-i square, denoted $E(d_i^u)$. We have

$$E(d_i^u) = (2c_2 + c_3) \cdot 2^i R,$$

where $2^i R$ is the side length of a level-i square, c_2 and c_3 are two constant factors representing the average random distance between two points in a pair of squares adjacent to each other or adjoin to each other on a corner, as shown in Figure 9 (right pane). Obviously, we have $c_2 \le \sqrt{5}$, and $c_3 \le 2\sqrt{2}$. Since updates are sent out at a rate of $\frac{v}{\delta}$, we have

$$\begin{aligned} E(C_m) &= \frac{v}{\delta} \cdot \sum_{i=1}^{H} \frac{(2 \cdot c_2 + c_3) \cdot 2^i \cdot R}{z} \\ &= \frac{v}{\delta} \cdot (2c_2 + c_3) \cdot \frac{R}{z} \cdot (2^{H+1} - 2) \\ &= O(v\sqrt{N}). \end{aligned}$$

Next, we consider the location query cost C_q. Based on the location query procedure we described above, the expected location query cost when A and B's MCS is level-i is

$$\begin{aligned} E(n_i^q) &= \sum_{j=0}^{i} \frac{E(d_i^u)}{z} \\ &= \sum_{j=0}^{i} \frac{c_1 \cdot 2^j R}{z} \\ &= (2^{i+1} - 1) \cdot c_1 \cdot \frac{R}{z}. \end{aligned}$$

Taking into account the various probability of the MCS being a level-i square, we have (for uniform data traffic pat-

tern),

$$\begin{aligned} E(C_q) &= \sum_{i=0}^{H} E(n_i^q) \cdot P_i^u \\ &\approx \sum_{i=0}^{H} (2^{i+1}-1) \cdot c_1 \cdot \frac{R}{z} \cdot \frac{3}{4^{H-i}} \\ &\approx 2^H \cdot c_1 \cdot \frac{R}{z} \\ &= O(\sqrt{N}). \end{aligned}$$

For localized data traffic pattern,

$$\begin{aligned} E(C_q) &= \sum_{i=0}^{H} E(n_i^q) \cdot P_i^l \\ &\approx \sum_{i=0}^{H} (2^{i+1}-1) \cdot c_1 \cdot \frac{R}{z} \cdot \frac{1}{2^{i+1}} \cdot \frac{1}{1-\frac{1}{2^{H-1}}} \\ &\approx c_1 \cdot \frac{R}{z} \cdot H \\ &= O(\log N). \end{aligned}$$

Finally, the storage requirement cost:

$$E(C_s) = \frac{N \cdot 3H}{N} = O(\log N).$$

□

4.2.2 DLM

As we described in Section 2, the DLM scheme uses multi-grained location information similar as HIGH-GRADE. A major difference is that in DLM the location servers of a node A are distributed uniformly across the network: one server in every level-K square, where K is a network parameter chosen between 1 and H. The choice of value for K involves a tradeoff between the location maintenance cost and the location query cost of the DLM scheme. Intuitive, when K is small, there is a location server for node A in every low level squares. Therefore, the location query cost is low as any node B can find a nearby server of A. However, the location update cost is high, since many servers need to be updated across the network. When K is large, the reverse is true. It is not surprising that the performance metrics we defined are functions of the parameter K, as shown in the following theorem.

THEOREM 9. *The expected location maintenance cost and the expected location query cost of the DLM scheme are:*

$$\begin{aligned} E(C_m) &= O(v(2^K + 2^{H-K}) + 2^{2K}) \\ E(C_q) &= O(2^{2K}). \end{aligned}$$

PROOF. The location update cost of DLM (denoted by C_{m1}) consists of two parts. First, when a node A moves out of its unit grid, it needs to update only its closes location server (the server within its level-K grid). In addition, when a node A makes a level-$(i+K)$ grid boundary crossing ($i \geq 0$), it needs to update all its servers within its level-$(i+K+1)$ grid.

We next derive the cost of updating all the servers within the level-$(i+K)$ grid. The DLM paper [10] did not discuss how to perform the updates. A naive approach would be to send a unicast message to each server. However, a better approach is to build a multicast-like tree structure that connecting all the servers within the level-$(i+K)$ grid, and send update messages along the "tree". Since there are 4^i servers in a level-$(i+K)$ grid, the tree will have $4^i - 1$ "edges". It is easy to see that each edge will have the geographical length of $2^K \cdot R$. Therefore, the total cost of updating all the servers within the level-$(i+K)$ grid is $2^K \cdot R \cdot (4^i - 1)$.

With same analysis used in HIGH-GRADE, the level-j grid boundary crossing rate is $P_j \cdot \rho = \frac{1}{2^j} \cdot \frac{\pi v}{2R}$. Therefore, the location update cost of DLM is

$$\begin{aligned} E(C_{m1}) &= \rho \cdot \frac{c_1 \cdot 2^K \cdot R}{z} + \sum_{i=0}^{H'} P_{i+K} \cdot \rho \cdot \frac{2^K \cdot R}{z} \cdot (4^i - 1) \\ &= 2^{K-1} \cdot c_1 \cdot \frac{\pi v}{z} + \frac{\pi v}{2z} \cdot (2^{H'+1} - 1) \\ &\approx (c_1 \cdot 2^{K-1} + 2^{H'}) \cdot \frac{\pi v}{z} \end{aligned}$$

Note that H' is the number of hierarchies over level-K, i.e., $H' = H - K$, H is the total number of hierarchies in the network.

In the DLM scheme, location server of node A is chosen by hashing the ID of node A to one unit grid in the level-K grid. If that unit grid is *void*, a "backup" search process is performed to search the entire level-K grid area in certain order. In addition, as the current location server moves, it needs to periodically search at least partial area of the level-K grid to ensure that it is still the "closest" node to the hashed position of node A; in case otherwise, it needs to pass the location information of node A to the "closer" node, which now serve as the new location server of A. In the worse case, the current server needs to search all the $2^K \times 2^K$ unit grids periodically to maintain the correctness of location server selection. Therefore, another part of location maintenance cost (denoted by C_{m2}) is

$$\begin{aligned} E(C_{m2}) &= \lambda_m \cdot \frac{R \cdot (2^K \cdot 2^K - 1)}{z} \\ &= O(2^{2K}), \end{aligned}$$

where λ_m is the rate of the periodical check by the current location server. Note that in the derivation, we assume the "backup" search process follows a tree like sequence similar as we discussed in the last subsection.

Therefore, $E(C_m) = O(v(2^K + 2^{H-K}) + 2^{2K})$.

The location query process of DLM is simple. To find the location of node A, a node B will query the location server of A within B's level-K grid. However, in the worst case, node B may need to search all the $2^K \times 2^K$ unit grids to find out node A's location server, because of the "backup" search scheme. Therefore, the location query cost of DLM is

$$E(C_q) = \frac{R \cdot (2^K \cdot 2^K - 1)}{z} = O(2^{2K}),$$

where λ_l is the number of queries performed on average by each node per second.

□

To balance the location maintenance and query cost, we usually minimize the sum of the two costs, which occurs when $K = \frac{H}{3}$. In that case,

$$E(C_m) = O(v\sqrt[3]{N})$$
$$E(C_q) = O(\sqrt[3]{N}).$$

We next present the storage requirement cost metric for the DLM scheme in the following theorem.

THEOREM 10. *The expected storage requirement cost of the DLM scheme is:*

$$E(C_s) = O(2^{2(H-K)}).$$

When $K = \frac{H}{3}$,

$$E(C_s) = O(\sqrt[3]{N^2}).$$

PROOF. In DLM, a node A has a location server in each level-K grid. This means that all the nodes in a level-K grid will host the location information for all the nodes in the entire network. Therefore, the storage requirement for each node is

$$E(C_s) = \frac{(2^H \cdot R)(2^H \cdot R) \cdot \gamma}{(2^K \cdot R)(2^K \cdot R) \cdot \gamma} = 2^{2(H-K)} = O(N^{2/3}).$$

Note that for the last line of the derivation, we assume $K = H/3$.

□

4.2.3 SLURP and SLALoM

In [8] and [9], the author gave the upper bound of the expected value of the location maintenance cost and location query cost for SLURP and SLALoM. The analysis of these studies are based on the same assumptions(e.g., γ is held constant, etc.) except that only the uniform traffic pattern is used. However, the results of their analysis still hold in the localized traffic pattern as we defined above, i.e., the results do not depend on which traffic pattern is assumed for the SLURP and SLALoM schemes. We duplicate their results in the following two theorems.

THEOREM 11 (WOO AND SINGH). *For the SLURP scheme,*

$$E(C_m) = O(v\sqrt{N})$$
$$E(C_q) = O(\sqrt{N}).$$

THEOREM 12 (CHENG ET AL.). *For the SLALoM scheme,*

$$E(C_m) = O(v\sqrt[3]{N})$$
$$E(C_q) = O(\sqrt[3]{N}).$$

The last theorem of our study is on the storage requirement cost of the SLURP and SLALoM scheme.

THEOREM 13. *For the SLURP scheme,*

$$E(C_s) = O(1).$$

For the SLALoM scheme,

$$E(C_s) = O(\sqrt[3]{N}).$$

PROOF. To analyze the storage requirement of SLURP, we note that in SLURP, the location information of a node A is stored in each node of the level-0 grid containing the hashed point of A. Assuming the node density is γ, we have

$$E(C_s) = \frac{N \cdot R \cdot R \cdot \gamma}{N} = O(1)$$

We next analyze the storage requirement for SLALoM. In SLALoM, every $K \times K$ unit squares area (a level-2 square) contains the location information of all N nodes in the network. Assuming $K = \theta(N^{1/3})$, we can calculate the number of nodes in the $K \times K$ unit squares area is

$$\frac{(N^{1/3} \cdot R)(N^{1/3} \cdot R)}{(A^{1/2} \cdot R)(A^{1/2} \cdot R)} \cdot N \propto \frac{(N^{1/3} \cdot R)(N^{1/3} \cdot R)}{(N^{1/2} \cdot R)(N^{1/2} \cdot R)} \cdot N \propto N^{2/3}$$

.

Therefore,

$$E(C_s) \propto \frac{N}{N^{2/3}} = O(N^{1/3})$$

□

4.3 Summary

We summarize the three performance metrics of the five location service schemes in Table 3, and make several observations about the results.

Not surprisingly, the results of the HIGH-GRADE scheme are most close to GLS, as their designs exhibit the most similarities. HIGH-GRADE and GLS have the same asymptotic

	HIGH-GRADE	GLS	DLM	SLURP	SLALoM
Location maintenance cost	$O(v \log N)$	$O(v\sqrt{N})$	$O(v\sqrt[3]{N})$	$O(v\sqrt{N})$	$O(v\sqrt[3]{N})$
Location query cost	$O(\sqrt{N})$ (uniform) $O(\log N)$ (localized)	$O(\sqrt{N})$ (uniform) $O(\log N)$ (localized)	$O(\sqrt[3]{N})$ (both)	$O(\sqrt{N})$ (both)	$O(\sqrt[3]{N})$ (both)
Storage overhead	$O(\log N)$	$O(\log N)$	$O(\sqrt[3]{N^2})$	$O(1)$	$O(\sqrt[3]{N})$

Table 3: Summary of three scalability metrics for the five location service schemes

location query costs and storage requirement costs. However, HIGH-GRADE has significant advantage in terms of the location maintenance cost. This is due to the usage of multi-grained location information. Consequently, HIGH-GRADE avoids frequent updates to remote servers, while keeping the local perimeter refreshing cost under control.

Comparing HIGH-GRADE and GLS in terms of total packet processing overhead—i.e., the sum of the location maintenance and query costs—HIGH-GRADE has several advantage. If the localized data traffic pattern is assumed, HIGH-GRADE has a better asymptotic results: $O(v \log N)$ vs. $O(v\sqrt{N})$ (results by summing the first two rows for the two schemes in Table 3). If the uniform data traffic pattern is assumed, then the two schemes have the same asymptotic cost: $O(v\sqrt{N})$. However, as we discussed in the previous sections, location query cost can often be reduced by various caching techniques. In HIGH-GRADE, since the bottleneck of the total cost is location query cost, any improvement from caching techniques will be reflected directly in the total cost. On the other hand, in GLS, since the location maintenance cost has higher asymptotic value, caching strategy may not bring much benefit.

We next observe that both SLALoM and DLM improve the total location maintenance/query costs over the SLURP scheme. However, they also introduce significant increase in the storage requirement cost. In the case of DLM, the storage requirement becomes a dominant constraint to the scalability. Another drawback of the SLALoM and DLM schemes is that, with preset K parameters, they do not take advantage of the localized data traffic patterns well. Their asymptotic bounds of location maintenance/query costs are the same for both uniform and localized traffic patterns.

Thus we conclude that the HIGH-GRADE scheme has balanced asymptotic results in all three scalability metrics analyzed. Specifically, when the data traffic pattern is highly localized, HIGH-GRADE achieves superior overall scalability in the order of $O(v \log N)$.

5. SIMULATION RESULTS

In this section, we compare the performance of HIGH-GRADE and GLS scheme. We focus on HIGH-GRADE and GLS because the analytical results for both schemes are similar. The GLS implementation we use for simulation is that of [7]. Our HIGH-GRADE simulation is implemented using the network simulator ns-2 [18][19]. Both simulations use IEEE 802.11 radio and MAC model as implemented in ns-2. Nodes are initially randomly placed across the entire network area using the uniform distribution. For all the simulation runs, the initial node density is 100 nodes/km^2. Therefore, the size of network area increases with the number of nodes. Nodes move according to the random way-

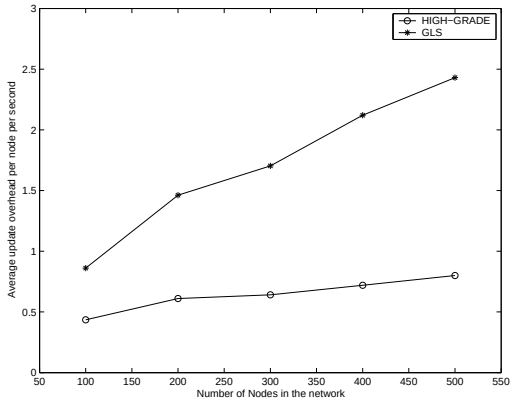

Figure 10: Average location update cost as a function of total number of nodes. Nodes move at speed up to 10 m/s.

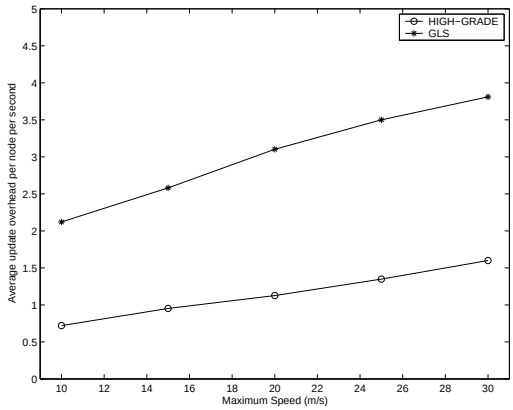

Figure 11: Average location update cost as a function of maximum node speed in a 400 nodes network.

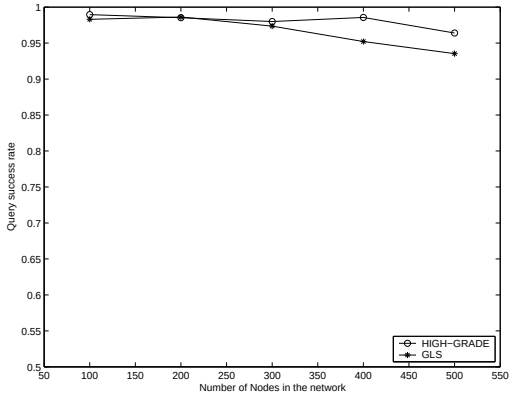

Figure 12: Query success rate as a function of total number of nodes. Nodes move at a maximum speed of 10 m/s.

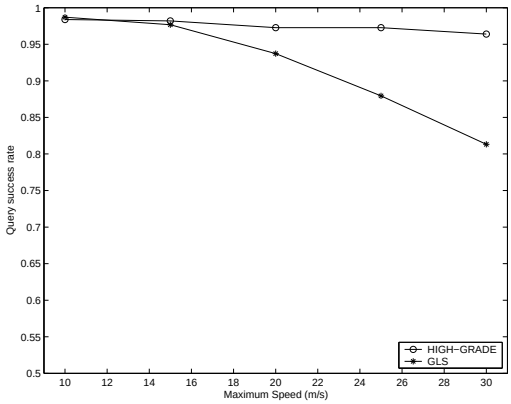

Figure 13: Query success rate as a function of maximum node speed in a 400 nodes network.

point model with no pause time. Each time a random target is chosen, a moving speed is selected between zero and a maximum moving speed. The maximum moving speed of the simulation is 10 m/s by default. The radio transmission range r_t for each node is 250 meters. Each simulation runs for 300 seconds, during which time, each node generates on average 15 location queries to random destination nodes. In the figures presented below, each data point is an average of five simulation runs. For both GLS and HIGH-GRADE, the side length of a level-0 square is set to be 250 meters, the transmission range. In all GLS simulations, location updating threshold is 100 meters.

Figure 10 shows the average location update cost as a function of N, the total number of nodes in the network. The location update cost (of a node) is defined as the number of location update packets that are originated at or are forwarded by a node per second. The average location update cost is the average over all nodes in the network. In HIGH-GRADE, nodes generate location update packets when they move across square boundaries, while in GLS, updates are generated periodically as a nodes moves more than the threshold distance. As we can see, the location update cost of both schemes exhibit sub-linear growth rate. In comparison, HIGH-GRADE grows much more slowly. This validates our analytic results, where the location update costs of the two schemes are $O(v \log N)$ vs. $O(v\sqrt{N})$.

Figure 11 shows the average location update costs as a function of maximum node speed for a 400 node network. Again, the simulation results confirm our analytic results, i.e., the costs grow linearly with the node speed for both HIGH-GRADE and GLS. This is because a higher node speed translates to higher boundary crossing rates in HIGH-GRADE and shorter update periods in GLS.

As we described, each node generates an average of 15 queries to random destinations over the simulation time. Figure 12 shows the query success rate—i.e., the total number of successful queries divided by the total number of queries in a simulation run—for HIGH-GRADE and GLS as a function of N. For HIGH-GRADE, a query may fail due to lost of update packets, or because location server(s) being queried has just moved into the LSP vicinity and has not yet obtained the location information. For GLS, query failure is

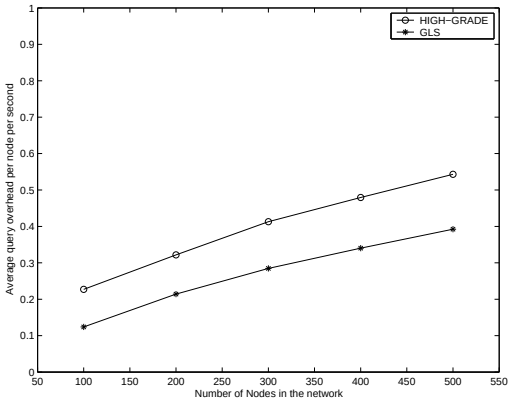

Figure 14: Average query overhead as a function of total number of nodes. Nodes move at speeds up to 10 m/s.

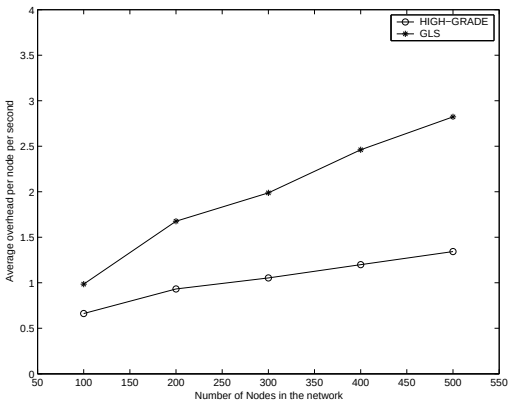

Figure 15: Average overhead as a function of total number of nodes. Nodes move at speeds up to 10 m/s.

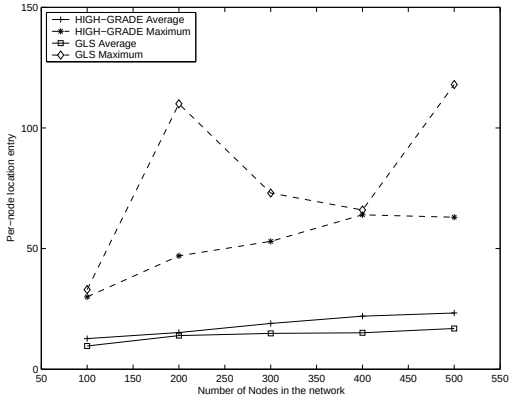

Figure 16: Storage requirement cost of HIGH-GRADE as a function of node number in the network with maximum node speed 10m/s.

due to stale location information stored on the servers. As we can see, both schemes maintain quite satisfactory query success rates.

Figure 13 shows the results for query success rate as a function of maximum node speed for a 400 node network. As we can see, when the speed is high, the query success rate remains high in HIGH-GRADE, but drops sharply in GLS. We speculate that this is due to the way location servers are updated in GLS. In GLS, when a node A needs to update one of its high level server, the update packet will be forwarded in many *steps*, each targeting a node in a level-i square (with increasing value of i). The choice of the correct next target in each step depends on the correctness of the location information in the current level-i square. Therefore, in GLS, the updates of lower level servers need to precede the updates of higher level servers, making the "convergence" time long. In face of high node speed, the GLS scheme is slow in adapting to node mobility, resulting in many stale location records on servers.

Figure 14 shows the average location query cost as a function of N. The location query cost (of a node) is defined as the number of query packets forwarded by a node per second. The average location query cost is the average over all nodes in the network. The results show that HIGH-GRADE and GLS have similar trends in their location query costs, with the HIGH-GRADE cost somewhat higher. We also observe that although the query rate is quite high (15 queries in 300 seconds for each node), the location query cost is much lower than the location update cost (comparing Figure 10 and Figure 14). Therefore the overall cost combining location update and query costs is dominated by the update part. Figure 15 shows the average of that overall cost for both HIGH-GRADE and GLS. It is clear that although HIGH-GRADE exhibits higher location query costs, its overall cost scales much better than GLS.

Finally, Figure 16 presents the storage requirement cost of HIGH-GRADE and GLS. For each simulation run, we calculate the average number of location records stored on each node at the end of the simulation time. In addition, we also find the maximum number of records a node stores. As we can see, the average storage requirement costs of the two schemes are close, both growing very slowly. This is in agreement with our analytic results, where both schemes has $O(\log N)$ asymptotic storage requirement costs. We also observe that GLS's average storage costs are slightly better than HIGH-GRADE's. However, the maximum storage costs of GLS are very erratic. We offer one possible explanation to this phenomenon. For both HIGH-GRADE and GLS, the network is in "perfect" hierarchy (i.e., $\frac{\sqrt{A}}{R}$ is a power of 2) when $N = 100$ and $N = 400$ in our simulations. In these cases, both HIGH-GRADE and GLS can distribute location server load uniformly across the network. In other cases, GLS distributes more load on the nodes located in the incomplete (i.e., not a power of 2) high level squares, resulting in high maximum storage cost.

6. CONCLUSIONS

In this paper, we have explored the design space of the location service scheme for ad hoc networks. We have classified several previously proposed schemes and discussed the tradeoffs of the key design choices. We have presented HIGH-GRADE, a new location service scheme which features a multilevel hierarchical structure and multi-grained location information. We have developed a uniform theoretical framework to compare the scalability of HIGH-GRADE with four other schemes. Our analysis shows that HIGH-GRADE has balanced costs in the three scalability metrics we defined, and has superior asymptotic bounds, especially when localized data traffic pattern is assumed. Our simulation experiments validate the theoretical analysis results. Based on these results, we believe HIGH-GRADE is an attractive choice in implementing location service for large ad hoc networks. In addition, we believe that our comparative study of the various location schemes will facilitate a deeper understanding of the issues related to the design of a scalable location service for ad hoc networks.

7. REFERENCES

[1] Onur Arpacioglu, Tara Small, and Zygmunt J. Hass. Notes on scalability of wireless ad hoc networks. http://www.ietf.org/internet-drafts/draft-irtf-ans-scalability-notes-01.txt, October, 2003.

[2] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[3] Charles Perkins and Elizabeth Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of Second Annual IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[4] Charles Perkins. *Ad Hoc Networking*. Addison Wesley Publishing Co., 2000.

[5] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking*, pages 243–254, 2000.

[6] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (dream). In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 76–84, 1998.

[7] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic

ad-hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 120–130, August 2000.

[8] Seung-Chul M. Woo and Suresh Singh. Scalable routing protocol for ad hoc networks. *Wireless Networks*, 7(5):513–529, 2001.

[9] Christine T. Cheng, H. L. Lemberg, Sumesh J. Philip, E. van den Berg, and T. Zhang. SLALoM: A scalable location management scheme for large mobile ad-hoc networks. In *Proceedings of Wireless Communications and Networking Conferencethe*, March 2002.

[10] Y. Xue, B. Li, and K. Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN '01)*, 2001.

[11] Sumesh J. Philip and Chunming Qiao. Poster: Hierarchical grid location management for large wireless ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 03), Poster session*, June 2003.

[12] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[13] Deniel Lewin. Consistent hashing and random tree: Algorithms for caching in distributed networks, Master Thisis, MIT, 1998. Available at the MIT library, http://thesis.mit.edu/.

[14] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[15] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[16] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. In *Mobile Computing and Networking*, pages 61–69, 2001.

[17] Steven R. Dunbar. The average distance between points in geometric figures. *College Mathematics Journal*, 28(3), may 1997.

[18] The VINT Project, The ns Manual, 2003. http://www.isi.edu/nsnam/ns/ns-documentation.html.

[19] CMU Monarch Group, CMU Monarch extensions to ns. http://www.monarch.cs.cmu.edu/.