# *LLS* : a Locality Aware Location Service for Mobile Ad Hoc Networks

Ittai Abraham
ittaia@cs.huji.ac.il

Danny Dolev
dolev@cs.huji.ac.il

Dahlia Malkhi
dalia@cs.huji.ac.il

School of Computer Science and Engineering
Hebrew University of Jerusalem
Jerusalem, Israel

## ABSTRACT

Coping with mobility and dynamism is one of the biggest challenges in ad hoc networks. An essential requirement for such networks is a service that can establish communication sessions between mobile nodes whose location is unknown. A *location service* for ad hoc networks is a distributed algorithm that allows any source node $s$ to know the location of any destination node $t$, simply by knowing $t$'s network identifier.

A location service has a *locality aware lookup* algorithm if the cost of locating destination $t$ from source $s$ is proportional to the cost of the minimal cost path between $s$ and $t$. A location service has a *locality aware publish* algorithm if the cost of updating the location service due to a node moving from $x$ to $y$ is proportional to the distance between $x$ and $y$.

In this paper we present LLS, the first location service for the Unit Disk Graph model whose lookup and publish algorithms have worst case locality guarantees and average case locality awareness efficiency for any source destination pair.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Distributed networks*

## General Terms

Algorithms, Theory.

## Keywords

Location Service, Ad hoc networks.

## 1. INTRODUCTION

In the widely used geometric ad hoc Unit Disk Graph model each node knows its location on the Euclidean plane and it can communicate with all other nodes whose distance is at most one unit. Consider the following natural question for mobile networks; A source node $s$ wants to initiate a communication session with a destination node $t$. The main problem is that node $s$ has no a priori knowledge of $t$'s current location. Thus, the first step in establishing a connection is to locate $t$'s whereabouts. Once the location of the destination is discovered the source can route messages and establish communication using well known geometric routing algorithms. A *location service for ad hoc networks* is a fundamental building block that allows any source $s$ to know the location of any destination $t$.

More generally, a geometric location service is useful in any Euclidean metric space: one needs to find the coordinates of targets in order to route toward them. Using geometric coordinates in general networks is made relevant not only by the ubiquity of GPS devices, but also by several recent techniques that embed internet nodes in a coordinate space. One of the pioneering mechanisms to predict network latency is based on the work of Ng and Zhang [26]. They embed the Internet latencies into a virtual geometric space (e.g., 3-D Euclidean) and characterize the position of any node with coordinates. The computed distances are used to predict the actual network distances. Following [26] other schemes have been developed to improve the embedding of internet hosts into virtual geometric spaces, e.g., [12], [33], [8], and [30]. Although the principles of LLS are applicable for general Euclidean spaces, the remainder of this paper focuses on describing LLS for ad hoc mobile networks.

In order to formally assess the utility of a geometric location service, we define formal measures of efficiency. Let the cost of a path be the sum of the costs of its edges. An important measure of a location service is its *lookup cost*: Given a source node $s$ and destination node $t$, the cost of locating $t$ from $s$ is the cost of the path induced by the location service until the location of $t$ known. A *locality aware lookup algorithm* is an algorithm whose lookup cost is proportional to the cost of routing between $s$ and $t$ when the location of the destination is known.

For mobile networks, whenever a node changes its location it must update the location service. Therefore, another important measure of a location service is its *publish cost*: Given that a node moves from location $x$ to a new location $y$, the publish cost is the total cost of the paths induced by the publish algorithm. A *locality aware publish algorithm* is one whose cost is proportional to the distance between the old location $x$ and the new location $y$.


*DIALM-POMC'04*, October 1, 2004, Philadelphia, Pennsylvania, USA.

## 1.1 Our Results

We present a Locality aware Location Service named LLS. Our location service is the first location service which has both worst case guarantees and average case efficiency. For worst case networks our lookup incurs a lookup cost of $O(d^2)$ for a source and a destination whose minimal cost path has length $d$.

For networks in which the expected ad hoc routing costs $\Delta$ times the distance from source to destination, LLS achieves in addition an average case linear cost over the distance, $O(d)$. Thus, in average case networks of nodes randomly positioned in the plane, where the routing cost is proportional to the distance from source to destination, lookup in LLS costs only a constant factor more than the distance between the source and the destination.

Our scheme is also the first to provide guaranteed average case efficient publishing. That is, our service ensures that the expected cost of updating the data structures due to a node's movement is bounded as a function of the distance of the movement. Specifically, when a node moves distance $d$, the average cost of publishing its new location is $O(d \log d)$.

The inherent locality of our scheme makes it fault tolerant both to node failure and to network partitions. When the network partitions, nodes within a connected component can locate each other because the location service is also collocated with them within the component. As for node failures, when some node containing location information fails, there is sufficient redundancy at incrementally increasing distances in the network to transparently make up for it.

## 1.2 Related work

*Geometric ad hoc routing.* The first step in our approach, is a routing algorithm for known geometric locations. The first routing algorithm to guarantee delivery is [19] (Kranakis et al.). Their face routing algorithm has no bound on the ratio between the cost of route and the cost of the minimal cost path. Both Bose et al. [6] (CGF) and Karp and Kung [17] (GPSR) propose an algorithm that combines greedy routing with face routing. These algorithms guarantee delivery and any source destination pair have expected cost $O(d)$, for average case networks, where $d$ is the distance between the source and the destination. The first algorithm that gives worst case guarantees is by Kuhn et al. [21]. They present a scheme in which, if the minimal cost path has cost $d$, then delivery with cost $O(d^2)$ is guaranteed, which is asymptotically optimal. In a follow up paper [22], they combine their bounded face routing with greedy routing to achieve a scheme that is both worst case asymptotically optimal and average case efficient. In our construction, we make use of an underlying geometric routing protocol, and assume both linear average case behavior and quadratic worst case.

*Location algorithms.* Location algorithms are measured by their lookup and publish costs. The basic location services studied by Camp et al. [7] either have an unbounded publish cost that may require to flood the whole network with updated location information (DLS, SLS) or have an unbounded lookup cost that may require to flood the whole network to find the destination (RLS). A standard technique for name services in wireless cellular networks (e.g., ISA-4 [1], GSM MAP [24]) employs a *home location register* (HLR) for each mobile host. A publish algorithm stores the whereabouts of a node at its home location. The lookup first routes to the home location, and from there to the current destination of the node. However, even this simple approach is challenging in a mobile ad hoc network, since there is no fixed infrastructure and the location servers themselves are dynamic.

One of the first approaches to address that in ad hoc networks, by Hubaux et al. [15], is to define the home of a node as a geometric area, and have all nodes in that area store location information. A similar solution for finding the home location is suggested in the context of sensor networks in the Geographic Hash Table (GHT) of [29]. In their approach, a home location is defined as a virtual coordinate. They enhance the underlying routing to reach the closest node to the virtual point. A similar concept is employed in the GeoQuorums of [10], where geometric coordinates determine the location of home servers. In GeoQuorums, these *focal point* coordinates define geographic areas that must be inhibited by at least one server at any time. The drawback of *all* of the home-based approach is that the cost of the lookup and of the publish may be arbitrarily high compared to the optimal path between source and destination.

In order to provide for better scalability and alleviate the problem of reaching specific location servers, several works suggest to replicate home location servers using quorum systems for availability and load balancing. Among these, the works of [28, 18, 13, 14] have no locality awareness. Other quorum based location services addressed locality in a partial way.

One of the early locality-aware location services that employs a hierarchy of partitions is provided for the general problem of object location in graphs by Awerbuch and Peleg [4]. Their solution does not make use of the geometric structure of ad hoc mobile networks, nor address their high dynamism. Consequently, their solution is not easily adaptable to dynamic mobile settings. In addition, their locality factors are somewhat large (polylogarithmic).

The approach taken by several works, e.g., in [31, 25, 5, 32], for quorum construction makes use of the planar structure of ad hoc networks. It defines a write quorum for updating location information of a node as a *column* of some choice trajectory, and potential some choice thickness. Similarly, a read quorum for querying location information is a row (of a choice trajectory and thickness). Trajectories are determined such that in average density networks, read and write quorums are likely to intersect. This method has good average case locality for lookup, but its publish cost is always the full diameter of the network, thus not proportional to the size of the movement. In addition, there are extreme cases in which read and write quorums might not intersect.

The *position based multi-zone* routing method of Amouris et al. [2] stores location information about each node in geometrically increasing discs, each disc referencing the smaller disc that contains the node. When a node moves a distance $2^i$, it broadcasts an update about the change to an area of radius $2^{i+1}$. Thus, both lookup and publish have locality awareness. The drawback of the scheme is that within a $2^i$ zone, location update is flooded to all nodes. This implies that each node in the network needs to maintain information (albeit not accurate) about every other node.

One of the pioneering works on efficient and scalable location services is by Li et al. in [23]. Similarly to the multi-zone method of [2], GLS utilizes a hierarchy of exponentially decreasing sets of regions (GLS uses squares rather than discs)

that cover the plane. Every node belongs to only $\log M$ squares (were $M$ is the diameter of the network). Using ingenious techniques drawn from the consistent hashing approach [16], every node has a designated hashed location server within each square, thus distributing the load of location services across the network. The path taken by a GLS lookup operation is bounded inside the minimal square that contains both the source and the destination.

Yet GLS does not achieve either of our goals, and supports neither worst case locality aware lookup, nor locality aware publish. There are several reasons for that, none of which is trivial to fix. First, their scheme makes little effort to proactively handle updates and out of date information. This problem arises when, for instance, a node crosses a grid boundary line. This causes a change in the role the node plays as a location server for others, performing this change of roles may cause the publish cost to be arbitrarily high compared to the distance taken. The authors state that indeed a remaining open question is improving the handling of node mobility. Second, there may be a source $s$ and destination $t$ that are arbitrarily close to each other, but the smallest square that contains both of them is arbitrarily large. As a consequence, even in average case networks, the cost of lookup does not have worst case bounds. Third, within each square GLS routes to a location server in order to find the target. In extreme network conditions when the network is sparse, routing to the location server, even if close by, could require worst case cost, while routing directly to the destination could be efficient. Thus, the worst case lookup cost could be arbitrarily high and have no worst case locality guarantees. This degraded performance in sparse networks was also observed by Guba and Camp [11].

A totally different approach focusing on worst case analysis is discussed in the Conclusion section of [21]. The authors describe an algorithm that we name the Iterative Bounded Flooding (IBF) algorithm. This algorithm runs in phases beginning with phase 1 and incrementing the phase by one until the destination is found. At phase $i$, the algorithm floods the network to all nodes whose minimal cost path from the source is at most $2^i$. IBF is asymptotically worst case optimal. Specifically, if the minimal cost of the path from source to destination is $d$ then IBF guarantees to reach the destination in cost $O(d^2)$. The main drawback of this approach is that its average cost is also $\Omega(d^2)$.

Recently, in the context of sensor networks and a slightly different model, Demirbas et al. [9], achieve $O(d \log d)$ move time and $O(d)$ find time.

### 1.3 Technical approach

For each destination node $t$, we define a virtual hierarchical cover of the $M \times M$ plane consisting of exponentially decreasing squares, whose origin depends on the node's id. Our solution is built incrementally in three steps, *Spiral*, *Spiral-Flood*, and *LLS*. In our basic Spiral algorithm, we publish $t$'s location on a spiral that spans increasingly large squares in the hierarchy, and likewise, search for $t$ in increasing spirals on the same virtual hierarchy. The lookup and publishing paths are guaranteed to cross at the first hierarchy level in which the squares containing the source and the destination intersect. This cover bears resemblance to the hierarchical grid of GLS [23]. However, we address mobility with techniques borrowed from GHT [29] by using virtual coordinates within the squares for storing information on $t$ rather than search for certain pre-designated nodes. This allows us to search in four grid squares around each point, and circumvent grid-boundary problems. Unlike GLS, our hierarchical cover forms a subsuming partition of the plane that is similar in spirit to the work on sparse partitions of Awerbuch and Peleg [3].

For most networks, this scheme suffices to have a lookup that is within an expected constant factor over the most optimal route. In order to further address worst case scenario, in our Spiral-Flood scheme we carefully interweave depth-bounded flooding stages with spiral lookup stages. This only increases the total cost by a constant factor, yet provides worst case quadratic location guarantee.

Finally, the full LLS scheme addresses publishing in the following way. First, we modify Spiral (or Spiral-Flood) so that within each lattice in the hierarchy, we publish a node's location not only in the square containing it, but also in the eight squares surrounding it. When a node moves within the nine square boundary, nothing happens. The stale information may eventually lead to a square that does not contain the node, but then one of the surrounding squares does. Location information in a lattice is updated only when the node leaves the nine-square boundary, at which time the distance traversed by the node (possibly over multiple steps) already exceeds the diameter of one square. As a consequence, we prove that the amortized cost of information updating due to a cumulative movement of distance $d$ is $O(d \log d)$.

In summary, our LLS scheme provides the first scheme which has both locality aware lookup and locality aware publish. Lookup is linear in average case networks, and is quadratic in the worst case, and publish is sub-quadratic.

## 2. MODEL AND NOTATIONS

Consider a set of $n$ nodes, $V$, that exists in the Euclidean plane $\mathbb{R}^2$. Each node $v$ has a unique name denoted $v.id$. Denote by $|uv|$ the $L_2$ distance between the location of the nodes $u$ and $v$. The underlying network is formed by a Unit Disk Graph $UDG = \langle V, E \rangle$ in which $u, v \in V$ have an edge $(u, v) \in E$ iff $|uv| \leq 1$.

We assume a nondecreasing cost function $c$, mapping edge lengths to real numbers. Formally, $c : [0, 1] \mapsto \mathbb{R}^+$, and for all $0 \leq x \leq y \leq 1$ we have $c(x) \leq c(y)$. This cost function abstraction generalizes the three common cost measures: hop count $c(x) = 1$, Euclidean distance $c(x) = x$, and energy $c(x) = x^\alpha$ for $\alpha \geq 2$. Given a path $p = u_1, u_2, \ldots, u_k$ such that $(u_i, u_{i+1}) \in E$ define the cost of the path to be $c(p) = \sum_{i=1}^{k-1} c(|u_i u_{i+1}|)$. Given two nodes $u, v$ let $d(u, v)$ denote the cost of the minimal cost path in $UDG$ from $u$ to $v$.

In order to be able to obtain worst case bounds on the cost of geometric routing we assume $\Omega(1)$ density. In this model there exists a minimum length $d_0$ such that for any network $\min_{u,v} |uv| \geq d_0$. Following the results in [20] it is also possible to remove the $\Omega(1)$ density assumption if the cost function, $c$, is linearly bounded. Formally $c$ is *linearly bounded* if there exists a constant $m$ such that $\forall x \in [0, 1] : c(x) \geq mx$. For linearly bounded cost functions it is possible to construct a connected dominating set backbone. Routing can be done on this backbone, which, by its nature, has bounded density. We note however that maintaining this backbone in a dynamic network incurs additional costs.

We assume all nodes are located inside a bounded square of size $M \times M$. Without loss of generality we normalize

the coordinates so that all nodes, $V$, are inside the square whose diagonal corners are $(0,0),(M,M)$. We further assume that each node knows its own location in the plane and its neighbors' locations.

## 2.1 Virtual Coordinates

A recurring issue in our scheme is the use of virtual coordinates that map to real nodes. The points are called virtual, since there is no guarantee that there is a node exactly at these points.

The natural way to map between nodes and virtual points is to partition the plane using a Voronoi diagram (see [27] for a survey on Voronoi diagrams and their applications). This partition maps each point $p$ in the square $M \times M$ to the node closest to $p$. For a node $x$ let $A(x)$ be the Voronoi polygon of $x$ (the area of all the points to which $x$ is the closest node). For any point $p \in \mathbb{R}^2$ let $\ell(p)$ be the closest node to $p$, thus, $p \in A(x) \quad \Leftrightarrow \quad \ell(p) = x$ .

Traditional geometric routing algorithms that combine greedy with perimeter routing [17, 22] guarantee that given the location of a node, the algorithm can route to it. These algorithms can be augmented as in GHT [29] so that given a virtual point $p$, the routing algorithm can reach the node $\ell(p)$ that is responsible for that virtual point. Their technique also adapts to node mobility. That is, when a node moves, the lines between its Voronoi polygon and its neighbors' Voronoi polygons changes. The nodes communicate and swap information about all the virtual points that cross boundaries due to these line changes.

In summary, there is a mapping between virtual points in the plane and the nodes responsible for them; and we can employ routing to virtual points that reaches the corresponding nodes. Hence, by a slight abuse of terminology, given a point $p$, we refer from here on to the node $\ell(p)$ simply as the node at $p$, or even the node $p$. And to the contrary, given a node $s$ we refer to the location of $s$ as the point $s$.

# 3. PROBLEM DEFINITION

We consider algorithms for the geometric location problem. In this problem there are two basic operations: Publish and Lookup. The *Publish*$(t.id; x, y)$ operation is called every time a node $t$ changes its location from $x$ to a new location $y$. It is also used in the first time a node joins the system, by executing *Publish*$(t.id; \perp, y)$. The *Lookup*$(t.id, s)$ operation is called by a source node $s$ in order to find location information on a destination node $t$ whose location is unknown.

We now define the complexity measure associated with locality aware location services. The first time a node joins, we would like to minimize the cost of its publish operation. For subsequent publish operations we would like the cost of publishing to be proportional to the distance taken. This is formally captured in the following definition.

**Definition** 3.1. *For a function $f$, a publish algorithm is* $f$-locality aware *if for any node $t$ with location $x$ that moves to a new location $y$ the expected cost of* Publish$(t.id; x, y)$ *is at most $f(|xy|)$.*

If the destination node does not exist we would like to minimize the cost of lookup until a failure result is returned to the source. Otherwise, we would like to have a locality aware lookup whose cost is proportional to the minimal cost path as defined below.

**Definition** 3.2. *For a function $g$, a lookup algorithm is* $g$-locality aware *if for any source $s$, and destination $t$, such that the minimal cost path costs $d = d(s,t)$, the cost of* Lookup$(t.id, s)$ *from source $s$ to destination $t$ is at most $g(d)$.*

Note that Definition 3.2 is a worst case bound that compares the lookup cost to the minimal cost path. Although it may not be possible for all networks, we would also like to get an average case bound on the cost of the lookup as a function of the *distance* between source and destination. The following definition captures this notion.

**Definition** 3.3. *For a function $g$, a lookup algorithm is* $g$-average case efficient *if for any source $s$, and destination $t$, the expected cost of* Lookup$(t.id, s)$ *from source $s$ to destination $t$ is at most $g(|st|)$.*

Certainly, building a lookup service with minimal cost can be done simply by storing complete location information at each node about all other nodes. The cost of an update, such as a node joining or moving, would be very high in this case. The goal is therefore to build a location service that simultaneously has the following properties: (1) a locality aware publish algorithm; (2) a locality aware lookup algorithm; (3) an average case efficient lookup on a large class of networks.

It is known from the lower bounds of [21] that even if $t$'s location is known, routing on *UDG*s may cost $\Omega(d^2)$, where $d$ is the cost of the minimal cost path. Simulations in [22] show that the cost of routing from source $s$ to destination $t$ is expected to be $O(d(s,t))$, which is also $O(|st|)$ for average case networks.

Our goal is to match these bounds for location services. Worst case lookup should cost $O(d^2)$, but for average case networks we aim to preform lookup with average cost that is only a constant factor more than the distance between the source and the destination, $O(d) = O(|st|)$. In case a node moves from location $x$ to destination $y$ we strive that the expected cost of the publish algorithm be a function of $|xy|$.

# 4. LLS ARCHITECTURE

We present our location service in a modular way. We decompose the scheme into three algorithms, each algorithm builds upon its predecessor's techniques and enhances it:

1. Spiral algorithm. This location service obtains a locality aware lookup algorithm for average case networks.
2. Spiral-Flood algorithm. This location service enhances the Spiral algorithm by obtaining locality awareness both for average case lookup, and for lookup in worst case networks.
3. LLS. This algorithm enhances the Spiral-Flood service with a locality aware publish algorithm.

## 4.1 Mapping to Hierarchical Lattices

In our construction, for each node identifier, we build a hierarchy of lattices associated with the node's identifier. The node's location is published on each lattice to the points closest to its location, and is looked up on lattice points closest to the location of the searching node.

Given a node $t$, we use a double index hash function $H(t.id) = \langle h_1(t.id), h_2(t.id)\rangle$ that maps node identifiers to coordinates inside $M \times M$. We denote $H(t.id)$ the *primary virtual home* of node $t$. Formally, $H : V \mapsto [0,M] \times [0,M]$.

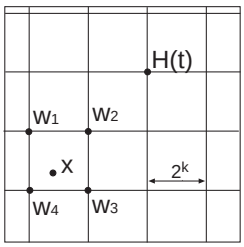


**Figure 1: Example of the lattice $L_k(t.id)$ and of $W_k(t.id, x) = \{w_1, w_2, w_3, w_4\}$**

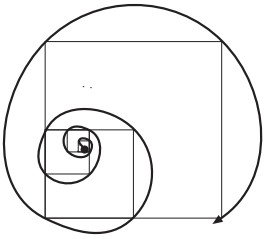

**Figure 2: The corners of all the tiles that cover the node are the set of virtual points to which location pointers are published. The spiral depicts a path that the *Publish* algorithm executes in order to reach the virtual points.**

For a node $t$, and a parameter $k \in \{0, 1, \ldots \log M\}$ we define $L_k(t.id)$ to be the lattice consisting of all lattice points $(h_1(t.id) + 2^k i, h_2(t.id) + 2^k j) \in M \times M$, for all $i, j \in \mathbb{Z}$. Alternatively, $L_k(t.id)$ can be thought of as the set of corner points of the tiling with squares of size $2^k \times 2^k$ having $H(t.id)$ as its origin.

For any point $x$ and tiling of size $2^k$ originating at the virtual home of $t$, $H(t.id)$, we define the *level-k location points of t from x*, $W_k(t.id, x)$, to be the set of the 4 corner points of the tile that covers $x$. Formally, for $k \geq 1$ we define $W_k(t.id, x) = \{w_1, w_2, w_3, w_4\}$ for every point $x$ and lattice $L_k(t.id)$, to be the set of the 4 lattice points that are closest to $x$ in the $L_\infty$ norm (with ties broken lexicographically), see Figure 1 for an example. When clear from the context, we will abuse notation and use $W_k(t.id, x)$ to denote the nodes that are responsible for the virtual points.

# 5. THE SPIRAL ALGORITHM

We begin by presenting the basic spiral algorithm. It achieves a locality aware lookup algorithm that performs as well as the underlying geometric routing algorithm. Intuitively, a node publishes its location information in a set of virtual points that form a virtual spiral that exponentially increases in distance. When a node initiates a lookup operation, it too performs a spiral like search path. The lookup finds the location of the destination when the two spirals intersect. The reason that these two spirals intersect is that the points in both spirals are computed relative to the same hierarchical grid whose origin is the primary virtual home of the destination.

The first time a node $t$ joins the network at location $y$, it invokes $Publish(t.id; \perp, y)$ to register its location. Whenever it moves from $x$ to $y$ in updates its location using $Publish(t.id; x, y)$. When a node $s$ wants to establish communication with node $t$, it issues $Lookup(t.id, s)$.

**Publish:** A node $t$ that moves from location $x$ to $y$ performs $Publish(t.id; x, y)$. This operation deletes the old location information (if there was any) and registers the new location $y$ of the node by inserting location information in $O(\log M)$ different virtual points.

Registering location information is done by storing *location pointers* in the auxiliary memory of the nodes in $W_i(t.id, y)$. Each location pointer contains two fields: the identifier of the destination and its location, in the form $\langle t, y \rangle$. Figure 2 depicts an example of the set of points to which information is published.

Deleting the information of the old location $x$ is done by visiting the respective nodes responsible for the old virtual points and deleting the information in them.

**Lookup:** A node $s$ that executes $Lookup(t.id, s)$ reaches the destination node $t$ by querying location information at the points associated with destination $t$, with increasingly larger distances from $s$. This process is very similar to the Publish operation. Once a location pointer is found, the information in the pointer is used to reach the destination.

## 5.1 Analysis

The lookup operation of the Spiral Algorithm is only as efficient as the underlying geometric routing that is used for routing to the nodes that are responsible for the virtual points. In this subsection we show that given a network with an efficient routing layer, the lookup operation costs only a constant factor more than the distance from the source to the destination.

In order to analyze the cost of the lookup operation in the Spiral Algorithm we first define the *locality awareness* of geometric ad hoc routing schemes in which the location of the destination is known.

**Definition** 5.1. *A routing protocol is* $\Delta$-locality aware *if given a source $s$ and point $y \in R^2$, the expected cost of routing from $s$ to the node at $y$ (more precisely, recall that this should actually reach $\ell(y)$, which is the node closest to $y$) is at most $\Delta|sy|$. Formally, let $r(s, y)$ be the cost of routing from $s$ to $y$ given that $s$ knows the approximate location $y$ then, $E_{s \in V, y \in R^2}\left[\frac{r(s,y)}{|sy|}\right] \leq \Delta$.*

In Section 10 we present simulation results showing that Greedy Face Routing is 2-locality aware for average case networks where nodes are uniformly distributed.

The following lemma shows that there is a location pointer at a distance that is proportional to the distance between source and target.

**Lemma** 5.2. *Let $\hat{k}$ be the minimal index such that $|st| = d \leq 2^{\hat{k}}$ then at least one of the nodes in $W_{\hat{k}}(t.id, s)$ contains a location pointer to node $t$.*

PROOF. Denote $t$'s location by $y$. Since destination node $t$ performed $Publish(t.id; y)$ then the nodes in $W_{\hat{k}}(t.id, y)$ contain location pointers towards $t$. Let $Q_1$ be the square of size $2^{\hat{k}} \times 2^{\hat{k}}$ that covers $t$ whose corners are $W_{\hat{k}}(t.id, y)$. Let $Q_2, \ldots, Q_9$ be the set of 8 adjacent squares of same size that have a joint edge or joint corner with $Q_1$.

Recall that we chose $\hat{k}$ such that $|st| = d \leq 2^{\hat{k}}$ therefore $s$ must be inside one of the squares $Q_1, \ldots, Q_9$. Thus when source $s$ performs the $\hat{k}$th phase of its lookup algorithm it will find a location pointer towards $t$.

Note that a similar argument holds for the degenerate case in which $t$ lies exactly on a lattice line or intersection. □

Hence for networks that have efficient routing, the cost of lookup is a linear function of the distance between source and destination.

**Theorem** 5.3. *For networks in which routing is $\Delta$-locality aware, for any source $s$ and destination $t$ the expected cost of locating $t$ is $O(|st|)$*

PROOF. Due to Lemma 5.2, the lookup algorithm will find a location pointer at phase $\hat{k}$, where $\hat{k}$ is the minimal index such that $|st| = d \leq 2^{\hat{k}}$. The expected cost of routing to all the points $W_1(t.id, s), \ldots, W_{\hat{k}}(t.id, s)$ can be bounded by the following expression: $\sum_{i \leq \hat{k}} 2 \cdot 4 \cdot 2^i \cdot \Delta = O(2^{\hat{k}}) = O(|st|)$ where the factor 2 is due to going to each of the virtual points and returning to $s$, 4 is due to the fact that there are 4 virtual points, $2^i$ is due to the maximal distance of the virtual points from $s$, and $\Delta$ is the overhead due to the underlying routing layer. Once a location pointer of level $\hat{k}$ is found, the expected cost of following the location pointers until $t$ is found can be similarly bounded by $O(|st|)$. □

# 6. THE SPIRAL-FLOOD ALGORITHM

The main drawback of the basic spiral algorithm is that it relies solely on the underlying routing algorithm for performance. In particular, there may exist extreme situations in which there is a low cost path from source to destination, but the cost of the minimal cost path from the source to the first virtual point is arbitrarily high.

The Spiral-Flood algorithm overcomes such cases. The algorithm combines the iterated bounded flooding (IBF) ideas from [21] for worst case bounding, with the average case efficiency of the Spiral algorithm.

In order to describe Spiral-Flood, we need to enumerate in a sequence the nodes visited in the Spiral algorithm. For a destination $t$ and a source $s$, the Spiral lookup operation *Lookup*$(t.id, s)$, coupled with the underlying geometric routing mechanism determine the sequence of nodes that are visited. We denote this sequence by *Spiral*$(t.id, s)$. Define *Spiral*$_j(t.id, s)$ to be the prefix path of *Spiral*$(t.id, s)$ containing the first $j$ nodes.

The Spiral-Flood lookup consists of phases, $i = 1 \ldots \log M$. When the accumulated cost of the Spiral lookup reaches $4^i$, we switch to flooding of depth $2^i$. This ensures that the cost of the combined algorithm is not more than a constant factor over the basic Spiral algorithm, and at the same time the cost does not exceed the worse case cost of the flooding algorithm. The pseudo code for the Spiral-Flood algorithm appears in Figure 3. The Publish algorithm is the same as the in basic Spiral algorithm.

*Lookup*$(t.id, s)$
Initialize *phase* $:= 0$
Until a location pointer for $t$ is found:

1. *Spiral stage:* Route on the path *Spiral*$(t.id, s)$ for $j$ hops, as long as the cumulative cost $c($*Spiral*$_j)$ is less than $4^{phase}$.
2. Return back to the source node $s$.
3. *Flooding stage:* Flood a search message to all nodes $r$ whose minimal cost path $d(s, r)$ is at most $2^{phase}$.
4. Converge the search results back to source node $s$.
5. Set *phase* $:=$ *phase* $+ 1$.

**Figure 3: The Spiral-Flood Lookup Algorithm. Node $s$ wants to establish a communication session with node $t$.**

## 6.1 Analysis

In this subsection we prove that the Spiral Flood algorithm worst case quadratic cost.

**Lemma** 6.1. *Spiral-Flood locates any destination $t$ from any source $s$ with cost $O(d(s, t)^2)$.*

PROOF. Denote $d = d(s, t)$, let $j$ be the minimal index such that $d \leq 2^j$. Clearly before the end of phase $j$ of the Spiral-Flood algorithm the destination will be found. This is true since the $j$th flooding phase would reach all nodes whose cost is a most $2^j$.

Given the $\Omega(1)$ assumption of a minimal distance between nodes (or any other bounded density assumption due to a CDS backbone), the sphere of nodes whose cost is at most $2^j$ has $O(2^{2j})$ nodes and hence also $O(2^{2j})$ edges. Each such edge is traversed at most 4 times. Therefore, the cost of a flooding during phase $j$ is $O(2^{2j})$.

By construction, the cost of the bounded phase-$j$ walk on *Spiral*$(t.id, s)$ is $O(4^i)$. Therefore, the total cost until the end of phase $j$ is $\sum_{0 \leq i \leq j} O(4^i) + (2^{2i}) = O(4^j) = O(d^2)$. □

Compared with the basic Spiral Algorithm, the Spiral Flood Algorithm incurs a constant factor overhead.

**Lemma** 6.2. *The cost of reaching the points $W_1(t.id, s)$ through $W_i(t.id, s)$ in the Spiral Flood Algorithm is at most a constant factor more than the cost of the basic Spiral Algorithm to reach the same set of points.*

The proof follows by noting that in each stage of the algorithm, the cost of the flooding stage is at most a constat factor more than the cost of the spiral search stage for that phase.

# 7. THE LLS ALGORITHM

In this section we describe our full scheme, LLS, in which proactive updating of location information due to node movements is proportional to the distance of the move. Formally we present a publish algorithm that is average case efficient. Achieving this aim is somewhat more involved than simply writing to all the virtual points in *Spiral*$(t.id, y)$ and deleting the points in *Spiral*$(t.id, x)$.

First, instead of publishing to only 4 virtual points at each level, our publish algorithm publishes to 16 virtual points. For any point $x$ and tiling of size $2^k$ originating at the virtual home of $t$, $H(t.id)$, we define $Z_k(t.id, x)$ to be the set of corner points of the 9 tiles of size $2^k \times 2^k$ that form a square $3 \cdot 2^k \times 3 \cdot 2^k$ whose center tile covers $x$. Formally we define $Z_k(t.id, x) = \{w_1, \ldots, w_{16}\}$ for every point $x$ and lattice $L_k(t.id)$, to be the set of the 16 lattice points that are closest to $x$ in the $L_\infty$ norm (with ties broken lexicographically). See Figure 4 for an example.

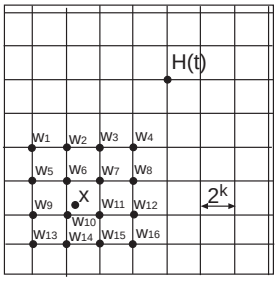


**Figure 4: Example of the lattice $L_k(t.id)$ and of $Z_k(t.id, x) = \{w_1, \ldots, w_{16}\}$**

Secondly, in order to reduce costs, a node does not update its level $k$ location pointers as long as it does not move a total of a certain distance proportional to $2^{k-1}$. This lazy update process is carefully done in a manner that still allows the lookup algorithm to be efficient and locality aware. Accordingly, we modify the location information stored at the publish nodes $Z_k(t.id, y)$ as follows. At the location nodes of $Z_i(t.id, y)$, we store information on a set of virtual points that defines the next hop towards the destination, in the form $\langle t, W_{i-1}(t.id, y)\rangle$. At $Z_0(t.id, y)$, we store the location $y$ itself.

**Publish:** A node $t$ that moves from location $x$ to $y$ performs $Publish(t.id; x, y)$.

1. If $W_0(t.id, x) = W_0(t.id, y)$ then no update is required.
2. Otherwise, do a spiral search, find the minimal index $i$ such that all four points of $W_i(t.id, y)$, have location information regarding $t$.
   (a) If $i = 0$ then no update is required. Otherwise, when $i > 0$:
      i. Using the location information stored in $W_i(t.id, x)$ (which contains $Z_{i-1}(t.id, x)$), recursively erase all the location information in the 16 points of each level $i-1, \ldots, 0$.
      ii. For $j = 1, \ldots, i-1$, store in $Z_j(t.id, y)$ the location pointer $\langle t, W_{j-1}(t.id, y)\rangle$.
      iii. Set the location pointers of level $i$ to point to the new location $\langle t, W_{i-1}(t.id, y)\rangle$.

Note that the minimality of the index $i$ implies that if $i > 0$ then the node should have moved out of the 9 tiles of level $i-1$. Thus, the node has moved a distance of at least $2^{i-1}$ since the last time the related nodes were updated. See Figure 5 for the LLS Publish algorithm.

**Lookup:** A node $s$ that executes $Lookup(t.id, s)$ follows the lookup algorithm as before, i.e., either Spiral or Spiral Flood. However, the search ends when a recursive location pointer is found of the form $\langle t, W(t.id, y)\rangle$ , where $y$ is the location of $t$.

Once such a node is found, repetitively use the *location pointers* in order to route from a point in $W_j(t.id, y)$ to a point in $W_{j-1}(t.id, y)$ and eventually for $j = 1$, to destination $t$.

At the lowest level, if the destination is not found at the square of the lowest level (of size $1 \times 1$), then a search is conducted in the 8 unit squares surrounding that square.

## 7.1 Analysis

*Publish*$(t.id; x, y)$

1. If all nodes $W_0(t.id, y)$ contain a location pointer for $t$ then RETURN.
2. Initialize $i := 1$
3. STEP A: Read all nodes in $W_i(t.id, y)$
4. If all nodes $W_i(t.id, y)$ contain a location pointer for $t$ goto STEP B.
5. Write in all 16 nodes of $Z_i(t.id, y)$ a location pointer $\langle t, W_{i-1}(t.id, y)\rangle$.
6. Set $i := i + 1$ and goto STEP A.
7. STEP B: Update the content of the 16 existing level $i$ location pointers to point to $\langle t, W_{i-1}(t.id, y)\rangle$.
8. For $j = i$ to 0 do
   Read all level $j$ nodes. Using the location pointers in the level $j$ nodes, route to all the level $j-1$ nodes and delete their location pointers.

**Figure 5: The LLS Publish Algorithm. Node $t$ updates location pointers once it arrived to a new location $y$.**

To address the complexity of the LLS publish algorithm we carefully study the movement of the node. Assume that the node performed $h$ publish operations, the *location history* of the node is the sequence $x_0, x_1, \ldots, x_h$ of locations. In order to bound the total cost of its $h$ publish operations we will keep track of the relative location of the intermediate nodes. We prove the bound on the cost in an amortized sense. The reason is that the node does not need to update its location pointers as long as it does not cross its 16-point boundary, which happens only if it moves a total of a certain distance. Once the node publishes its location, the cost of publishing is offset by the total distance it has covered since its last update.

Our analysis focuses on average case networks, and in particular, assumes that the network has $\Delta$-locality aware routing, Let $d(X_j)$ denote the total distance moved by the node in its first $j \le h$ hops, thus $d(X_j) = \sum_{i=1}^{j-1} |x_i x_{i+1}|$. For any level $k$, let $c_k(x_j, x_{j+1})$ denote the cost associated with writing and deleting location pointers on level $k$ virtual points as a result of $Publish(t.id; x_j, x_{j+1})$. Denote the cost of updating the level $k$ virtual points due to the first $j$ location changes as $c_k(X_j) = \sum_{i=1}^{j-1} c_k(x_i, x_{i+1})$. Our goal is to bound the total cost $c(X_j) = \sum_{i=1}^{\log M} c_i(X_j)$ as a function of total distance $d(X_j)$.

**Lemma** 7.1. *If* Publish$(t.id; x_j, x_{j+1})$ *changes level $k$ location pointers then the expected cost of this change is bounded by $E[c_k(x_j, x_{j+1})] \le \Delta \cdot 2^{k+6}$.*

Proof. When the $Publish(t.id; x_j, x_{j+1})$ changes level $k$ location pointers, it deletes old level $k$ pointers and writes to its new ones. The expected cost of each of these two operations can be bounded by $\Delta \cdot 2 \cdot 16 \cdot 2^k$ .

Where the factor $2 \cdot 16 \cdot 2^k$ is due to the total distance of a spiral path visiting all 16 points and returning and $\Delta$ is the overhead due to the underlying routing layer. □

When level $k$ location pointers are not changed despite the move, there is a potential gain towards future updates. This is formally stated in the following Theorem.

**Theorem** 7.2. *The total expected cost of publishing of location history $x_1, \ldots, x_h$, is bounded by $E[c(X_h)] \le O(d \log d)$ where $d = d(X_h)$.*

PROOF. While moving, the node occasionally updates its location pointers. Consider a specific level $k$. A node moving from $x_{j-1}$ to $x_j$ updates its level $k$ pointers when the location of its level $k-1$ pointers at $x_{j-1}$ differ from its new location's ($x_j$) level $k-1$ pointers. Formally, let $i_1 < i_2 < \cdots < i_m$ be the sequence of all indices such the level $k$ location pointers were updated when the node reached $x_{i_\ell}$ for $1 \le \ell \le m$. Due to our publish algorithm, the distance the node moves between any two updates of level $k$ pointers is at least $2^{k-1}$. That is, for any such index $i_\ell$ we have $\sum_{i_{\ell-1}<j\le i_\ell} |x_{j-1}x_j| = d(X_{i_\ell}) - d(X_{i_{\ell-1}}) \ge 2^{k-1}$. Thus the number of updates $m$ is bounded by $d(X_h)/2^{k-1}$. From Lemma 7.1, the expected cost of the update of level $k$ pointers is $\Delta \cdot 2^{k+6}$. Therefore, by summing on the whole sequence, $E[c_k(X_h)] \le \Delta \cdot 2^{k+6} m \le \Delta \cdot 2^7 \cdot d(X_h)$. The expected total cost of publishing the updates during the location history $x_1, \ldots, x_h$, is the sum of the costs over all levels that were updated during the total move. Thus, the cost is $E[c(X_h)] \le \Delta 2^7 \cdot d(X_h) \log d(X_h)$. If $d(X_h) \ge M$ then the expected cost may be bounded by $\Delta 2^7 \cdot d(X_h) \log M$. □

# 8. FAULT TOLERANCE

Our location service is inherently fault tolerant both to network partitions and to node failures.

In case of a network partition, nodes within the vicinity of a detached node remain able to locate it and communicate with it. More precisely, let a set of nodes $S$ detach from $V \setminus S$. For a node $t \in S$, let $k$ be the maximal index such that $W_k(t.id, t) \subseteq S$. Then all of the nodes within distance $2^k$ can locate $t$ and communicate with it. In fact, even in extreme cases where a partition forms weird shapes, for any $\ell > k$ such that we have $W_\ell(t.id, t) \cap S \neq \emptyset$, far nodes in $S$ that have lookup spirals intersecting these location points are also able to locate $t$.

As for node failures, our scheme provides immediate fallback due to the multiplicity of location points that store information about any node. In particular, when a node $s$ searching for $t.id$ encounters a faulty location point, it simply skips it and moves on to the next higher level. This guarantees location at the first non-faulty level that is greater than the distance from the source and the target. That this works follows from the following fact. If the minimal index at which intersection occurs between $W_k(t.id, t)$ and $W_k(t.id, s)$ is $k$, then higher level location points intersect as well.

As a final frontier for fault tolerance, the flooding stage will always locate the target (if it is connected) at a quadratic cost.

Additional fault tolerance is provided by employing the perimeter replication techniques of [29], by storing all of the information belonging to virtual point $x$ also on all the nodes surrounding its perimeter.

# 9. IMPROVING LOCALITY AWARENESS

Assume a network in which routing is $\Delta$-locality aware. A natural question to ask is whether on top of the locality-aware routing mechanism, our Spiral paradigm can provide location services with costs arbitrarily close to minimal. In addition to its theoretical value, this question has real-life motivation: For nodes that incur frequent lookups, we would like to drive the lookup cost down as much as possible.

In this section, we focus on optimizing the cost of the lookup operation. We obtain a lookup scheme whose cost is $(1+\varepsilon)\Delta$ on top of the actual distance to target. The decreased lookup cost is obtained at the cost of increasing the publish cost by a constant factor. We could do the reverse in a similar manner, if the frequency of updates is expected to exceed that of lookups.

In order to reduce the cost of lookup, denote by $B(u, r)$ the ball around node $u$ with radius $r$. Let us define $W_{k,\ell}(t.id, x)$ to be $L_k(t.id) \cap B(x, 2^\ell)$; that is, the set of lattice points in $L_k(t.id)$ within distance $2^\ell$ from $x$. The trick will be to slightly increase the range of publishing into $W_{k,k+\rho}$, for a parameter $\rho$ of the construction determined below; and to query only one node in $W_{k,k}$. More precisely, we now define publish and lookup as follows:

**Publish:** For any $i \in \{0, 1, 2, \ldots, \log M\}$, location pointers on $t.id$ located at $y$ are stored in the auxiliary memory of the nodes in $W_{i,i+\rho}(t.id, y)$, where the parameter $\rho$ is determined below. A node $t$ that moves from location $x$ to $y$ performs $Publish(t.id; x, y)$ by deleting old information and storing new information in the nodes described above.

**Lookup:** A node $s$ executes $Lookup(t.id, s)$ by performing the following loop. Until $t$ is found, for $i = 0, 1, 2, \ldots, \log M$, read from the closest node in $W_{i,i}(t.id, s)$ until a node is found that has a location pointer $\langle t, W(t.id, y)\rangle$, where $y$ is the location of $t$. Once such a node is found, repetitively use the *location pointers* in order to route from a point in $W_j(t.id, y)$ to a point in $W_{j-1}(t.id, y)$ and eventually for $j = 1$, to destination $t$.

Locality awareness stems from the following lemma:

**Lemma** 9.1. *Denote $|st| = d$. Let $\hat{k}$ be the minimal index such that $2^{\hat{k}} + d \le 2^{\hat{k}+\rho}$. Then the $\hat{k}$'th lookup step from $s$ finds a location pointer to node $t$.*

PROOF. The lemma follows from the fact that $B(s, 2^{\hat{k}}) \subseteq B(t, 2^{\hat{k}+\rho})$. Hence, every node in $W_{\hat{k},\hat{k}}(t.id, s)$ is contained in $W_{\hat{k},\hat{k}+\rho}(t.id, t)$. Therefore, every node in $W_{\hat{k},\hat{k}}(t.id, s)$ holds a location pointer to node $t$. □

As a consequence of this lemma, the cost of lookup from $s$ to $t$ is bounded by $\sum_{i\le\hat{k}} 2^i\Delta \le 2\cdot 2^{\hat{k}}\Delta$. Bearing in mind that $\hat{k}$ was the first index for which $2^{\hat{k}} + d \le 2^{\hat{k}+\rho}$, we now simply need to set $\rho = \rho(\varepsilon)$ to be $\rho \ge \log(1 + \frac{8\Delta}{\varepsilon}) = O(\log(\varepsilon^{-1}))$, in order to obtain that the total lookup length is at most $(1+\varepsilon)\Delta d$.

We now get to the increased cost of publishing. We only sketch the difference here due to space limitations. In our original Spiral method, publishing a node's new location was done in the 4 corner points of the tile covering the node. We now change that to include all the lattice-$k$ points within distance $2^{k+\rho}$ from the node. Their count is at most $2^{2\rho}$, and the total cost of reaching them is proportional to $2^{k+\rho}/2^k$. Therefore, the increased cost is a constant factor over our original scheme, where the constant depends on $\rho = \rho(\varepsilon)$.

We also briefly comment on how to complete our scheme for locality-aware updates. This requires maintaining an invariant that all lattice-$k$ points within a ball of radius $2^{k+\rho}$ surrounding a node have up-to-date information about the interior ball containing it. In order to prevent frequent updates, we initially publish within a ball with double this radius, i.e., a ball of radius $2^{k+1+\rho}$, and update only when the

invariant is broken. This again incurs only a constant-factor increase over the original LLS publish costs.

In summary, at the cost of a constant factor increase in update costs, we obtain a location service whose cost is almost as low as the cost of the underlying routing infrastructure.

## 10. SIMULATIONS

In this section we present simulation results for the LLS scheme. Following the observations of [22] about the importance of the network density on routing performance, we tested our scheme in randomly generated graphs with varying densities. The density of a graph is defined relative to a unit disk as follows, a random graph on a $T \times T$ square with density $\delta$ has $\delta T^2/\pi$ nodes.

We tested the underlying Greedy Face Routing algorithm to virtual points, and the LLS Lookup algorithm. Our measurements were performed on Unit Disk Graphs that were generated by randomly placing nodes on a square with 15 units side length.

**Routing.** For the underlying greedy face routing algorithm, for each density $\delta$, we generated 1200 random graphs for $\delta \leq 8$ and 200 random graphs for $\delta > 8$, for each graph, we randomly chose 50 sources nodes, for each source $s$ we chose a random point $p$ on the square and found the closest node $\ell(p)$ to the virtual point $p$. We measured the following parameters for each density:

1. Connectivity: The percentage of node pairs $s, \ell(p)$ that were connected on the Unit Disk Graph.
2. Cost/Distance: The average ratio between the cost of routing from a source $s$ to a destination $\ell(p)$ and the Euclidean distance between $s$ and $p$.

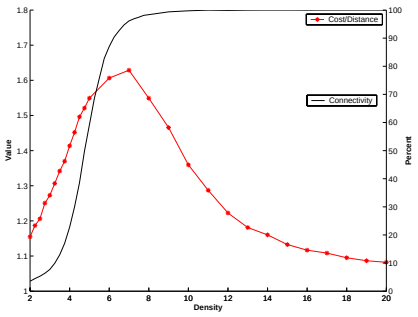


**Figure 6: Connectivity percentage and Cost/Distance values for greedy face routing from a random source to the node closest to a random point.**

Our results in Figure 6 show that the average cost of routing on a random graph to the node closest to a virtual point is only a small constant factor times the Euclidean distance between the node and the virtual point.

**Lookup.** For each density value $\delta$, we generated 500 random graphs for $\delta \leq 10$ and 100 random graphs for $\delta > 10$, for each graph, we randomly chose 10 sources nodes, for each source we randomly chose a destination node. We measured the following parameters for each density:

1. Cost/Distance: The average ratio between lookup cost and Euclidean distance between source $s$ and target $t$.
2. Cost/Shortest path: The average ratio between lookup cost and the minimum cost path from $s$ to $t$ on the UDG.
3. Connectivity: The percentage of $s, t$ pairs that were connected.
4. Flood: the percentage of cases in which the lookup algorithm had to use flooding in order to find the destination.

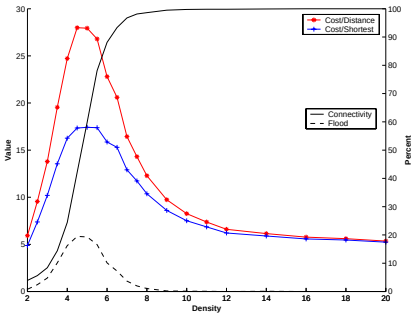


**Figure 7: Measurements for Lookup algorithm: Cost/Distance and Cost/Shotest path values, Connectivity and Flood percentages.**

Our results in Figure 7 show that lookup is locality aware, the average cost of locating a mobile node is only a constant factor from the optimum. For low densities, the low stretch factor may be attributed to the fact that the connected components are relatively small. Both Cost/Distance, Cost/Shortest, and Flood ratio peak around density 5. In this critical density, the graph is mostly connected but finding short paths to route on is still hard.

**Publish.** We measured the cost updating the location pointers due to a random hop. The distance moved in each hop was a varied by a parameter $\ell$. For each hop length $\ell$, we generated 100 graphs, for each graph we chose 20 sources and for each source we performed a random hop, with a randomly chosen angle and a randomly chosen length in $[\ell, \ell + 1/2)$. For every value of $\ell$, we measured the average cost of updating location pointers due to the random hop over all sources and graphs.

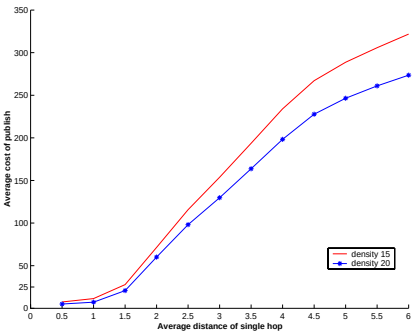


**Figure 8: Average cost of publish relative to the average length of the hop for graphs with densities 15 and 20.**

Our results in Figure 8 show that the cumulative cost of updating location pointers due to a random walk is proportional to the length of the average hop. This result agrees with our theoretical bounds of $O(d \log d)$.

## 11. CONCLUSIONS

This paper presents LLS, the first location service for mobile ad hoc networks to guarantee both worst case bounds and average case efficiency. Our scheme has inherent fault tolerance both for node failures and for network partitions. The generalization of our construction results in a geometric location service that is locality aware in any Euclidean metric space.

For any network the worst case cost of our lookup is quadratic $O(d^2)$. For average case networks LLS achieves linear $O(d)$ expected lookup cost. Mobility of nodes is handled in an efficient proactive manner. LLS ensures that the expected cost of publish is $O(d \log d)$. We provide simulation results for the LLS scheme that conform our theoretical bounds.

Our simulations show that routing in random ad hoc networks is 2-locality aware for various densities. It would be interesting to give a formal proof backing this result.

A potential area of improvement is reducing the asymptotic cost of the publish algorithm. The worst case cost of the publish algorithm can be easily bounded using similar techniques as in our lookup algorithm. An open question is whether an expected $O(d)$ is achievable for *both* publish and lookup.

## Acknowledgments

The authors would like to thank Marina Shudler for simulations. Fabian Khun and the anonymous referees provided helpful comments.

## 12. REFERENCES

[1] Cellular radio telecommunication intersystem operations. Technical Report Technical Report TIA/EIA-41-D, Telecommunications Industry Association, Washington, DC.

[2] K.N. Amouris, S. Papavassiliou, and M. Li. A position-based multi-zone routing protocol for wide area mobile ad-hoc networks. In *Proceedings of the IEEE Vehicular Technology Conference (VTC)*, pages 1365–1369, 1999.

[3] B. Awerbuch and D. Peleg. Sparse partitions. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.

[4] B. Awerbuch and D. Peleg. Online tracking of mobile users. *Journal of the ACM*, 42(5):1021–1058, 1995.

[5] I. Aydin and C. C. Shen. Facilitating match-making service in ad hoc and sensor networks using pseudo quorum. In *Proceedings of 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, 2002.

[6] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[7] T. Camp, J. Boleng, and L. Wilcox. Location information services in mobile ad hoc networks. In *Proceedings of the IEEE International Conference on Communications (ICC '02)*, 2002.

[8] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, distributed network coordinates. In *HotNets Workshop*, 2003.

[9] M. Demirbas, A. Arora, T. Nolte, and N. Lynch. Brief announcement: Stalk: a self-stabilizing hierarchical tracking service for sensor networks. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 378–378. ACM Press, 2004.

[10] S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC 2003)*.

[11] N. Guba and T. Camp. Recent work on gls: a location service for an ad hoc network. In *Proceedings of the Grace Hopper Celebration (GHC '02)*, 2002.

[12] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002)*, 2002.

[13] J. L. Welch H. Lee and N. H. Vaidya. Location tracking with quorums in mobile ad hoc networks. *Ad Hoc Networks*, 1(4):371–381, 2003.

[14] Z. J. Haas and B. Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2):228–240, 1999.

[15] J. P. Hubaux, J. Y. Le Boudec, and M. Vetterli Th. Gross. Towards self-organizing mobile ad-hoc networks: the terminodes project. *IEEE Comm Mag*, 39(1):118 –124, January 2001.

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

[17] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM Press, 2000.

[18] G. Karumanchi, S. Muralidharan, and R. Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Symposium on Reliable Distributed Systems*, pages 4–13, 1999.

[19] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11 th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[20] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *Proc. $22^{nd}$ ACM Int. Symposium on the Principles of Distributed Computing (PODC)*, 2003.

[21] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proc. of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications (Dial-M)*, pages 24–33. ACM Press, 2002.

[22] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In *Proc. $4^{th}$ ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc)*, 2003.

[23] 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 '00)*, pages 120–130, August 2000.

[24] M. Mouly and M. Pautet. *The GSM System for Mobile Communications*. 1992.

[25] B. Nath and D. Niculescu. Routing on a curve. In *HotNets-I, Princeton, NJ*, 2002.

[26] E. Ng and H. Zhang. Predicting internet network distance with coordiantes-based approaches. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM '02)*, 2002.

[27] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000. 671 pages.

[28] R. Prakash and M. Singhal. Dynamic hashing + quorum = efficient location management for mobile computing systems. In *Proceedings of ACM Symposium on Principles of Distributed Computing*, 1997.

[29] S. Ratnasamy, B. Karp, Y. Li, F. Yu, R. Govindan, S. Shenker, and D. Estrin. GHT: A geographic hash table for data-centric storage. In *The First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, October 2002.

[30] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM '03)*, San Francisco, CA, 2003.

[31] I. Stojmenovic. A scalable quorum based location update scheme for routing in ad hoc wireless networks. Technical Report TR-99-09, SITE, University of Ottawa, 1999.

[32] J. Tchakarov and N. Vaidya. Efficient content location in wireless ad hoc networks. In *IEEE International Conference on Mobile Data Management (MDM)*, 2004.

[33] S. Wilbur, T. H. Saleem, B. M. Pias, and J. Crowcroft. Lighthouses for scalable distributed location. In *The 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.