3DLS: Density-Driven Data Location Service for Mobile Ad-Hoc Networks

Roy Friedman  Noam Mori
Computer Science Department
Technion - Israel Institute of Technology
Haifa, 32000
Israel

Email: {roy, noam}@cs.technion.ac.il

Abstract

Finding data items is one of the most basic services of any distributed system. It is particularly challenging in ad-hoc networks, due to their inherent decentralized nature and lack of infrastructure. A data location service (DLS) provides this capability. This paper presents 3DLS, a novel density-driven data location service. 3DLS is based on performing biased walks over a density based virtual topography. 3DLS also includes an autonomic dynamic configuration mechanism for adapting the lengths of the walks, in order to ensure good performance in varying circumstances and loads. This is without any explicit knowledge of the network characteristics, such as size, mobility speed, etc. Moreover, 3DLS does not rely on geographical knowledge, its decisions are based only on local information, it does not invoke multi-hop routing, and it avoids flooding the network. The paper includes a detailed performance study of 3DLS, carried by simulations, which compares 3DLS to other known approaches. The simulations results validate the viability of 3DLS.

Keywords: Ad-hoc networks, data location service, virtual topography.
1 Introduction

Context of this Study: Mobile Ad-Hoc Networks (MANETs) are formed by a collection of mobile nodes, each equipped with wireless communication capabilities, without relying on any fixed infrastructure or centralized administration. In order to maintain network connectivity, each node may act as an ad-hoc router, forwarding data packets for other mobile nodes that may not be within direct transmission range of each other.

The ability to locate a specific data item in a MANET is required for many of its envisioned applications. Consider for example the following scenario. An increasing number of mobile phones these days are equipped with WiFi (in addition to cellular communication). This could potentially be used to implement the following two applications:

1. Collaborative caching of Internet content [10, 27, 31]. In this application, we turn the collective local caches of the browsers on each phone into a larger cache. Thus, if a user tries to access a certain URL, it first checks if the URL is locally cached inside the MANET. If yes, the URL is fetched locally, thereby saving cellular bandwidth.

2. Decentralized ad-hoc school-wide file sharing. In this application, students can share various files such as photos, music, and video clips with each other. This, without consuming expensive cellular bandwidth, and while enjoying the relatively high bandwidth of WiFi. Yet, to implement such an application, users need to be able to search for files.

Data Location Services (DLSs) provide this capability, and are the focus of this paper. A DLS typically has two methods, one for publishing (or registering) data items and the other for looking up the data. Publishing the data may involve updating local data structures and sending out advertisement messages. On the other hand, looking up data items may involve inspecting local data structures and sending out lookup messages. The number of advertisement and lookup messages generated by a DLS implementation impact the efficiency of the implementation. Additionally, due to the nature of MANETs, the way these messages are sent is also important. For example, in a MANET, flooding the entire network is very resource consuming. Also, multi-hop routing is very expensive, since route discovery and route maintenance are costly [23, 30], especially when the nodes are moving and the topology changes. Moreover, broadcast messages are more costly than point-to-point messages (despite the medium being a broadcast medium at the physical layer). This is due to the link layer protocol and the fact the broadcast messages are typically transmitted at significantly lower bandwidth (1 or 2 Mbps) compared to point-to-point ones (up to 54 Mbps) [11].

When examining existing DLS implementations [1, 17, 19, 20, 29], including attempts to utilize Distributed Hash Tables (DHT) over MANETs [4, 8, 24, 32], it is evident that they exhibit at least one of the following shortcomings:

- An occasional need to flood the network with data advertisement/lookup messages or other types of messages, which generates a significant amount of traffic.

- The use of several multi-hop routing steps on each data advertisement/lookup. This might result in very long physical paths and requires periodic maintenance of routing paths. As we mentioned above, routing is a very expensive operation in MANETs.

- The reliance on geographical knowledge, which demands additional hardware (e.g., GPS). Accurate geographical knowledge is not always available, e.g., in most laptops, inside buildings, etc.

- Lack of support for search queries that incorporate regular expressions.
A high replication degree of data location information, which increases the memory usage of the protocol.

In this paper, we present a novel density-driven DLS that overcomes these limitations. In particular, our service never floods the network, does not employ multi-hop routing nor relies on any geographical knowledge. It supports range queries and requires a very low replication level. Yet, it is highly reliable and efficient.

Contributions of this Work: We start by presenting a new density-driven substrate that can be defined on top of MANETs, which we refer to as virtual topography. In order to construct the virtual topography, each node locally computes its dynamic virtual height according to the density of its current surroundings. The denser the surroundings of a node is, the higher is its virtual height. Yet, our notion of density refers to the number of paths in the neighborhood of a node rather than the number of nodes. The significance of this will be revealed later in the paper.

The combination of the nodes and their virtual heights forms a dynamic density-driven virtual topography. However, each node only knows its own virtual height and the virtual heights of its immediate neighbors and none of the nodes knows larger portions of the entire topography, which is important for scalability and efficiency. The virtual topography can then be used to direct density biased walks in a greedy local manner to the same relatively small set of nodes, whose virtual heights is a local maxima. We refer to these nodes as hilltops. We show empirically that the number of hilltops does not depend on the number of nodes in the network, which contributes to the scalability of our protocol. The virtual topography can be built and maintained solely by using the known heartbeat mechanism. We show how to do that and analyze some of this virtual topography’s appealing properties.

We then present 3DLS, a Density-Driven Data Location Service that utilizes the density-driven virtual topography substrate. The idea behind 3DLS is that every data item in the network is published in a number of hilltops. This is done efficiently using a density biased walk on top of the virtual topography. Similarly, data is looked up in a sufficient number of hilltops, using the same density biased walk, to ensure that lookup walks are very likely to visit one of the hilltops in which the corresponding data was published. In other words, hilltops serve as rendezvous points for publications and lookups. In addition, we employed a mechanism that keeps the published information at the hilltops in face of constant topological changes in the network.

3DLS does not suffer from any of the shortcomings listed above. Moreover, it is very robust, it is tolerant to nodes’ mobility and it is extremely lightweight in terms of the traffic it induces and the replication level it requires. 3DLS includes a self tuning mechanism that autonomously adjusts the length of the advertisement/lookup walks to the properties (such as size and speed of movement) of the network on which it is running. Interestingly, these self-adjustments are performed without building an explicit knowledge of the network properties, but rather only by observing their local symptoms. As mentioned before, 3DLS refrains from flooding the network, does not invoke multi-hop routing, and does not rely on geographical knowledge.

Finally, we measure the performance of our protocol using simulations and compare our protocol to other representative DLS implementations. These simulations validate our approach and our protocol design choices.

---

1 Notice that when data items are small, typically they are the ones being located by the DLS. However, if the data items are large, then the DLS only provides a mapping to the data owner, in which case accessing the data itself may require an additional routing step, regardless of the implementation of the DLS.
Paper's road-map: The model and basic assumptions are listed in Section 2. Section 3 presents the virtual topography and its properties, while the 3DLS protocol appears in Section 4. The simulations results are described in Section 5. We discuss related work in Section 6 and present our conclusions in Sections 7.

2 System Model and Definitions

In this work we focus on wireless mobile systems. Specifically, we assume a collection of nodes placed in a given finite size area. A node in the system is a device owning an omni-directional antenna that enables wireless communication. For the purpose of the protocol description, we assume the simplified uniform transmission disk model. That is, a transmission of node \( p \) can be received by all nodes within a disk centered on \( p \) whose radius depends on the transmission power, referred to in the following as the transmission disk. The radius of the transmission disk is called the transmission range. The combination of the nodes and the transitive closure of their transmission disks forms a wireless ad-hoc network.

We assume that all of the nodes in the network share a similar transmission range, meaning, if node \( q \) is located within the transmission disk of node \( p \) than, with high probability, \( p \) is symmetrically located within the transmission disk of \( q \). In such a case \( p \) and \( q \) are called direct neighbors of each other and will receive each other's transmitted messages. We also state that each node is a direct neighbor of itself.

In the following, \( N(1, p) \) refers to the set of direct neighbors of node \( p \) and \( N(k, p) \) refers to the k-hop neighborhood of node \( p \) (including itself). By considering \( N(1, p) \) as a relation (defining the set \( N(1, p) \)), we say that a node \( p \) has a path to a node \( q \) if \( q \) appears in the transitive closure of the \( N(1, p) \) relation. In such a case we define the distance in hops between nodes \( p \) and \( q \) as \( \text{Min} \{ k \in \mathbb{N} | q \in N(k, p) \} \).

As nodes can physically move, there is no guarantee that direct neighbors at time \( t \) will remain direct neighbors at a later time \( t' > t \). Additionally, messages can be lost. For example, if two nodes \( p \) and \( q \) transmit a message at the same time, then if there exists a node \( r \) that is a direct neighbor of both \( p \) and \( q \), then \( r \) will not receive either message, in which case we say that there was a collision. New nodes may join the network and existing nodes may leave at any time, either gracefully or by suffering a crash failure. Nodes that crash or leave the network may rejoin it later.

We assume that no geographical knowledge is available to any of the nodes, namely the nodes do not know their positions or their speed vectors. Finally, we assume that each node has a unique identifier; we denote the identifier of node \( p \) by \( \text{ID}_p \).

In any Data Location Service (DLS) protocol, we refer to every atomic piece of information (e.g., a web page, a file, etc.) as a data item. We refer to the nodes that share their hosted data items as data owners. Each data item is stored entirely in its data owner’s memory (or any other attached storage device). Every data item is identified using a short string called a key (e.g., a URL address, a file name, etc.). We refer to the nodes that want to retrieve a data item as clients. The DLS protocol handles data location queries initiated by clients. Every data location query contains a description of the requested data item’s key.

When the DLS stores mappings from data items to data owners, as typical for large data items, the DLS replies to the client with the ID of a data owner hosting a data item who’s key matches the requested key, if one exists. After a client receives a reply from the DLS, it may contact the data owner in order to retrieve the data item itself. We do not consider this last procedure of retrieving the data item itself as part of the DLS. On the other hand, if the DLS stores the actual data, which may occur with small data items, then the

---

2The assumption of a symmetric transmission disk greatly simplifies the definition of neighbors, and thereby also the description of the protocol. Yet, all our simulations are performed in a realistic model with distortions, background noise, etc. Hence, this assumptions does not restrict our work.
content of the data item is sent in the reply. In this case, the client does not need to contact any additional node.

3 The Density-Driven Virtual Topography

Recall that the purpose of the density-driven virtual topography is to function as a substrate on which it would be relatively easy to direct many density biased walks to the same relatively small set of nodes. The virtual topography is achieved by assigning each node a virtual height. The height of node \( p \), which we denote by \( H_p \), is derived directly from its current density using the following formula:

\[
H_p = \sum_{q \in N(1,p)} |N(1,q)|
\]  

(1)

This formula tries to best reflect the notion of nodes’ density in the absence of geographical knowledge and it differs from simply using the size of their 2-hop neighborhoods \(|N(2,p)|\) as their density. The latter approach only counts the number of nodes in the 2-hop neighborhood of node \( p \). Our approach counts the number of different 2-hop routes that exist from node \( p \) to any of the nodes in its 2-hop neighborhood.

For example, in Figure 1, the physical density of node \( r \) in (b) is obviously greater than its physical density in (a) even though \(|N(2,r)| = 5\) in both cases. However, when applying Formula (1) to Figure 1(a), we get \( H_r = 9 \), matching the observation that node \( r \) is placed in a sparse surrounding. Yet, in Figure 1(b), where the physical density of node \( r \) is obviously greater, \( H_r = 17 \). This example shows that our formula better reflects the notion of density among nodes than simply using the size of their 2-hop neighborhoods as their density.

Having defined the height of each node as a measure of its density, we can further define an order relation “higher than” on a set of nodes as a lexicographical order of their tuples \( <H,ID>\), meaning that \( p \) is higher than \( q \) if and only if \((H_p > H_q \lor (H_p = H_q \land ID_p > ID_q))\).

Considering the virtual topography, we notice that nodes that are placed in a relatively dense part of the network form what can be viewed as a hill. Similarly, nodes that are located in a relatively sparse part of the network form what can be viewed as a valley. We define a hilltop to be a node that is currently higher than all of its direct neighbors (excluding itself). Each hilltop uniquely defines one hill. At every given time we associate a single hilltop, and hence, a single hill, to each node recursively as follows: the hilltop of a hilltop node is itself and the hilltop of a node that is not a hilltop is the hilltop of its highest direct neighbor. We denote the hilltop of node \( p \) by \( Hilltop_p \). Intuitively, \( Hilltop_p \) is the node to which node \( p \) would eventually reach if it would follow a path greedily choosing the highest direct neighbor at each step until reaching the local maxima. Each node constantly keeps track of its current hilltop.
Figure 2: The network’s topology (a) translates into its virtual topography (b)

<table>
<thead>
<tr>
<th>Name</th>
<th>Denoted</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID&lt;sub&gt;p&lt;/sub&gt;</td>
<td>The unique ID of node p</td>
</tr>
<tr>
<td>Height</td>
<td>H&lt;sub&gt;p&lt;/sub&gt;</td>
<td>The current height of node p</td>
</tr>
<tr>
<td>Hilltop</td>
<td>Hilltop&lt;sub&gt;p&lt;/sub&gt;</td>
<td>The current hilltop of node p</td>
</tr>
<tr>
<td>Direct neighbors</td>
<td>N(1, p)</td>
<td>The unique IDs of all of the direct neighbors of node p</td>
</tr>
</tbody>
</table>

Figure 3: The structure of node p’s heartbeat packet

In Figure 2, we show an example of how a network’s topology translates into its virtual topography. In (a) we see a network’s topology; the numbers inside the nodes represent the number of direct neighbors they have. In (b) we see the network’s virtual topography, which contains one hill. The numbers inside the nodes and their brightness represent their height (darker is higher). The arrows represent each node’s highest direct neighbor. The black node (marked 35) is the hilltop.

### 3.1 Virtual Topography Construction And Maintenance

In order to construct and maintain the virtual topography we use the common heartbeat mechanism. Each node broadcasts a heartbeat message every constant time (the structure of such packets is presented in Figure 3) and maintains a neighbors list based on the received heartbeat messages. The last heartbeat message received from each direct neighbor is attached to each neighbor in the neighbors list. Each node removes neighboring nodes from its neighbors list if sufficient time has passed since it last received their heartbeat broadcast. Upon any change to its neighbors list, namely, receiving a heartbeat message or removing a neighbor from the list, each node recalculates its current hilltop using the algorithm presented in Figure 4. Each node recalculates its own height (Formula (1)) according to the information within its neighbors list just before broadcasting each of its heartbeat messages. These hilltop and height recalculations are done asynchronously and independently by each of the nodes.

<sup>3</sup>For the purpose of creating and maintaining the virtual topography it would have been sufficient to include in the heartbeat message only the number of direct neighbors node p has - |N(1, p)|. However, the complete list of IDs of all of the direct neighbors is required for the density biased walk forwarding algorithm described in Section 4.2.1.
Upon any Neighbors\_List\_p, change do
(1) highest\_nbr := p;
(2) Hilltop\_p := p;
(3) foreach q ∈ Neighbors\_List\_p do
(4) if (q is higher than highest\_nbr)
(5) highest\_nbr := q;
(6) endforeach;
(7) Hilltop\_p := Hilltop\_{highest\_nbr};

Figure 4: Hilltop calculation algorithm (as preformed by node p)

Due to mobility and topological changes, the virtual topography of the network is constantly changing. A hilltop p can suddenly realize that one of its direct neighbors, node q, is higher than itself, either by receiving q's heartbeat containing its current higher height or by recalculating its own lower height. This event is referred to as a hilltop switch. As a result of these topographical changes, two hills can suddenly merge to form one wider hill and a hill can suddenly split into two separate hills.

Notice that the knowledge that a node has regarding its current hilltop usually suffers from propagation delays. In case a hilltop switch occurs between nodes p and q, nodes that had p as their hilltop and are not direct neighbors of p and q might still believe that their hilltop is p after q had taken over the hilltop position. These hilltop misconceptions can occur because there is no special communication induced by such events. These nodes will have an updated knowledge regarding their current hilltop once the information regarding the hilltop switch will be propagated to them through their neighbors' periodical heartbeats, which could take some time depending on the heartbeat frequency and the distance in hops between them and p or q. It is not unreasonable that while the update is propagated to the edges of the hill another hilltop switch may occur resulting in a constant hilltop misconception at the edges of the hill. Although it is an important phenomenon to be aware of, the protocol we describe in this work easily overcomes it, as described in Section 4.

3.2 Virtual Topography Properties

As stated above, the purpose of the density-driven virtual topography is to function as a substrate on which it would be relatively easy to direct many density biased walks to the same relatively small set of nodes. These relatively small set of nodes are the hilltops, which, as described in detail in Section 4, serve all of the nodes in the network as data location servers. The following properties of the virtual topography, mainly the physical distribution and the number of hilltops in the network, which are exemplified in Figure 5, are important in order to support the data location service protocol. The significance of these properties in reference to the 3DLS protocol is further discussed in Section 4.

The following properties are empirical and have been observed throughout extensive simulations using the JIST/SWANS simulator\(^4\). The network characteristics used in the simulations are presented in Figure 6.

1. There is a relatively small number of hilltops per area: As a direct result of the formula we use to calculate the nodes' height (Formula (1)) and our system model (Section 2), the difference between the heights of two nodes will become negligible as they grow physically closer to each other. This means that the nodes' height changes gradually across the network and sharp changes are extremely rare. This property gives the virtual topography the resemblance to a real moderate topography containing

\(^4\)http://jist.ece.cornell.edu/
Figure 5: The virtual topography of a MANET consisting of 200 nodes with 250m transmission range placed uniformly at random on a 2000m X 2000m area. The black nodes are the hilltops and the dashed circle around each of them represents its transmission range.

a relatively small number of hilltops per area. In our simulations, the average number of concurrent hilltops in the network was 31.43 (out of 1,000 nodes). As a counter example, if a completely random value would have been used for the height of each node, a very jagged topography, which would contain a relatively large number of hilltops per area, would have been formed. In our simulations, the average number of concurrent hilltops in such a random jagged topography was 103.63.

Empirically, by extensively testing a wide range of network settings, the average number of hilltops can be described by the following formula: \( \#\text{Hilltops} \approx \frac{A}{(\pi r^2 \cdot 3.2)} \) where \( A \) is the test area and \( r \) is the transmission range. In particular, the number of hilltops does not depend on the number of nodes in the network, which contributes to the scalability of our protocol. Also, the average area covered by a hill is 3.2 transmission disks. I.e., in networks where the average degree of the nodes is 10, only \( 1/32 \approx 3\% \) of the nodes are hilltops. Explaining why the number of hilltops is linear to the ratio between the test area and the area of one transmission disk, and why the linear coefficient is 3.2
<table>
<thead>
<tr>
<th>Network Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Area</td>
<td>1400X1400 m²</td>
</tr>
<tr>
<td>Node count</td>
<td>1000</td>
</tr>
<tr>
<td>Transmission range</td>
<td>80m (each node has 10 neighbors in average)</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>5.4Mb/sec</td>
</tr>
<tr>
<td>Mobility model</td>
<td>Random Walk (each traveled leg is set to 200m)</td>
</tr>
<tr>
<td>Speed</td>
<td>Ranging from 0.5 – 5m/s</td>
</tr>
<tr>
<td>Mac protocol</td>
<td>IEEE 802.11 with no message fragmentation</td>
</tr>
<tr>
<td>Routing protocol</td>
<td>NONE</td>
</tr>
</tbody>
</table>

Figure 6: The default network characteristics used in the simulations

![Partitioning of test area into zones](image)

Figure 7: The partitioning of the test area into zones.

is left for future work.

2. Hilltops are distributed uniformly across the network. In order to verify this hypothesis we have conducted an experiment measuring the uniformity of the hilltops’ distribution. We then compared the experiment’s results to the expected results when the hilltops are indeed known to be distributed uniformly across the network. This comparison has been done using the well known statistical $\chi^2$ test indicating that our hypothesis is valid.

For this, we have run a simulation using the network characteristics described in Figure 6. During the simulation, which lasted for one simulated hour, we have taken a snapshot of the topography of the network every 30 simulated seconds (a total of 120 snapshots). The collected snapshots are denoted by $Snapshot_t$ ($1 \leq t \leq 120$). The number of hilltops in $Snapshot_t$ is denoted by $\#Hilltops_t$. We have partitioned the test area of each snapshot into 9 equal size zones as illustrated in Figure 7, which are denoted by $Zone_i$ ($1 \leq i \leq 9$). The number of hilltops in $Snapshot_t$ that are located inside $Zone_i$ is denoted by $\#Hilltops_t(i)$. If our hypothesis is in fact true, $\forall t \forall i \#Hilltops_t(i) \approx \frac{\#Hilltops_t}{9}$. We have tested our hypothesis using the $\chi^2$ test for each snapshot separately. A formal description of the $\chi^2$ test used for analyzing each $Snapshot_t$ is presented in Formula (2).
\[ \chi^2 = \sum_{1 \leq i \leq 9} \left( \frac{\#Hilltops_t(i) - \frac{\#Hilltops_t}{9}}{\frac{\#Hilltops_t}{9}} \right)^2 \]  

(2)

The smaller the \( \chi^2 \) test results, the more uniformly the hilltops' distribution is. The average \( \chi^2 \) over all of the snapshots taken throughout the simulation is 3.19. The average \( \chi^2 \) is greater than zero since there are only an average 31.37 hilltops in each snapshot. In addition, the number of hilltops in each zone is an integer while the expected number of hilltops in each zone is \( \frac{\#Hilltops_t}{9} \), which typically is a float.

In order to get an additional evidence that our \( \chi^2 \) test results are in fact an indication of the uniformity of the hilltops' distribution, we have conducted an additional experiment, in which we checked the \( \chi^2 \) value of randomly selected nodes. That is, the second experiment is similar to the first, but instead of analyzing the location of the hilltops at each snapshot, we have analyzed the location of \( \#Hilltops_t \) randomly chosen nodes at each \( Snapshot_t \). Meaning, if in some snapshot there were \( X \) hilltops, we have randomly chosen \( X \) nodes (from all of the nodes) and analyzed their distribution across the network zones. We have applied the same \( \chi^2 \) test over the location of the randomly chosen nodes in each snapshot and averaged the result over all of the snapshots taken throughout the simulation. The average \( \chi^2 \) result in this last experiment was 8.48. The reason this number can be larger than 0 is that we are taking a small discrete sample on each snapshot.

According to these results, the likelihood that the hilltops' distribution is in fact uniform is higher than the likelihood that the same number of uniformly random chosen nodes distribute uniformly across the network. Hence, it is probabilistically safe to conclude that the hilltops are indeed distributed uniformly across the network. One may ask how come the \( \chi^2 \) test results when the hilltops' locations were analyzed were better than when the randomly chosen nodes' locations were analyzed. This is because hilltops cannot be direct neighbors of each other by definition while randomly chosen nodes do not have such a restriction. In addition, due to the formula we use to calculate nodes' height (Formula 1), the probability that two hilltops will be physically close to each other is lower than the probability that two randomly chosen nodes will be physically close to each other.

3. Once a hilltop switch occurs, the switching hilltops are most likely to be direct neighbors of each other. Empirically, the average distance in hops between them is about 1.1 hops (reported as handoff length in Figure 14(b) below). This property is important in order to minimize communication during the data handoff protocol described in Section 4.2.3 and to limit most hilltop misconceptions to a single hop error.

There are many protocols with different purposes (e.g., routing, broadcasting, etc.) in which nodes require knowledge regarding their direct neighbors [7]. Many of these protocols also utilize heartbeats in order to obtain direct neighbors awareness among their nodes. Our topography maintenance can share the same heartbeat mechanism with these other protocols with the only overhead of slightly increasing the heartbeat packet size. As we have shown, the density-driven virtual topography is fairly easy to construct and maintain, and its cost is minimal whenever a heartbeat protocol is already active in the network. In the following section, we show how to utilize these properties in order to implement an efficient Data Location Service.
4 The Density-Driven Data Location Service Protocol

In the proposed 3DLS protocol no single node constantly takes more responsibilities than others. Nodes may assume any of the following roles, more than one if required. A Data Owner is a node that hosts one or more data items that may be required by other nodes in the network. Data owners are responsible for advertising their hosted data items at a large enough number of hilltops. A hilltop (Section 3) is a node that for a short period of time, as long as it is a hilltop, serves as a data location server. Hilltops are responsible for providing timely and efficient responses for query messages regarding the location of advertised data items. A Client is a node that request data items on behalf of an application or any higher layer by issuing a query message to a large enough number of hilltops.

In this section, we provide the design description of 3DLS. We start by giving a brief overview of the protocol followed by a detailed description.

4.1 Protocol Overview

In 3DLS, a data owner periodically advertises the data items it currently hosts to a certain number of hilltops starting with its own hilltop by following a density biased walk as described in Section 4.2.1. Such advertisements utilize the density-driven virtual topography in order to pass through a “sufficient” number of hilltops using a relatively short walk. Hilltops along this walk cache the data location information received in the advertisement message before forwarding the message to the next node on the walk. Nodes along the walk that are not hilltops forward the advertisement message without caching its content.

Recall that due to nodes’ mobility, a hilltop node \( p \) might undergo a hilltop switch and adopt another node \( q \) as its new hilltop. Node \( q \) could have already been a hilltop back when \( p \) was a hilltop or it could have become a hilltop during the hilltop switch. In any case, in order to preserve the data location information at the current hilltops, an orderly handoff must take place between \( p \) and \( q \). During the handoff, \( p \) should transfer the data location information from its cache to \( q \) using a data handoff message that propagates via a density biased walk. Fortunately, one of the properties of the virtual topography (Section 3.2) is that the expected distance in hops between \( p \) and \( q \) in these cases is close to 1.

A client may locate any data item on the network by sending out a query message. The query message is similarly propagated along a density biased walk, passing through a number of hilltops until it either finds the location of the requested data item or until the number of visited hilltops along the walk is large enough to assume that the item does not exist, as described below. In case the requested data item has been located at some hilltop along the walk, that hilltop will respond to the query message with a reply message which will be sent back to the client along the reverse path.

During every density biased walk, as well as the reverse path taken by the reply, the data location information stored in the visited hilltops’ caches is piggybacked on top of the propagated message and cached when the message reaches the following hilltops. When a hilltop piggybacks the content of its cache on top of any message, it first has to subtract from the expiration field of each ad the time elapsed since it first received the ad in order to maintain the expiration’s consistency. This process helps disseminate data location information across multiple hilltops.

We have optimized the piggybacking mechanism in order to avoid redundant data items. When a message is supposed to carry an ad about a data item already piggybacked by this message, only the ad with the furthest expiration time is kept and the other ad is discarded. This optimization does not change the success rate of the protocol. However, in most cases the client will only receive a single ad as a response to its query message, whereas without this optimization, a client could have received multiple ads as a reply for the same
Name | Description
--- | ---
Last heartbeat packet | The last heartbeat packet that has been broadcasted by the forwarding node - added in order to prevent loops
#Visited Hilltops | The number of hilltops visited along the walk
Route Back | A stack of IDs of the nodes which are only visited once along the walk - used for backtracking
Trail | A set of IDs of the nodes visited along the walk and their direct neighbors

Figure 8: The navigational information inside every density biased walk message

query. In any case, the client receives the ad with the furthest expiration time among the ads it would have received prior to this optimization.

4.2 Protocol Details

In this section, we provide details regarding the design of 3DLS. We first describe in detail the density biased walk used by all of the protocol elements: data advertisement, data handoff, and data lookup. We then proceed to describe each of the other protocol elements separately.

4.2.1 Density Biased Walk

The density biased walk is used whenever a node needs to send a message of any type (advertisement, handoff or lookup) to its hilltop and possibly to other hilltops as well. The forwarding algorithm utilizes information received from the virtual topography substrate and information collected along the walk in order to navigate efficiently from hill to hill across the network. The main objective of the density biased walk is to pass through a large enough number of hilltops using a minimum number of hops. The walk is stopped according to a dynamic stopping condition referred to as a walk stopper. There are 3 types of walk stoppers (described in detail in Section 4.2.5) matching each of the messages’ types:

1. Advertisement stopper - used as a stopping condition for data advertisement messages
2. Handoff stopper - used as a stopping condition for data handoff messages
3. Lookup stopper - used as a stopping condition for data lookup messages

All of the transmissions along the density biased walk are point-to-point transmissions, meaning, the packet is transmitted to a specific destination node (one of its direct neighbors), which is the only node to receive the transmission. This is in contrast with a broadcast transmission in which all the nodes that reside within the transmission disk of the transmitting node receive the transmission. As we mentioned in the introduction, at least for the 802.11 protocol (WiFi), point-to-point messages are more efficient than broadcasts.

As the message passes along the density biased walk, calculating its next hop as it goes, it gathers information needed in order to navigate efficiently within the virtual topography. This information is referred to as Navigational Information and presented in Figure 8. The navigational information includes the last heartbeat packet, the #Visited_Hilltops, the Route_Back and the Trail. The Trail is a set of IDs of the nodes visited along the walk and their direct neighbors.
In case there is a need to limit the packets’ size along the walks, a few relaxations can be made to the definition of the Trail with the intent of reducing its memory consumption. For example, by using a probabilistic data structure like a bloom filter in order to store the Trail [5].

The forwarding algorithm is executed by a node once it receives a density biased walk message of any type. The first phase of the algorithm (presented in Figure 9 as getHighestNbr) strives to guide the walk towards the hilltop of the forwarding node, unless that hilltop had already been visited along this walk. This is done by forwarding the message to the highest direct neighbor of the forwarding node and by that greedily climbing to the nearest hilltop using the steepest climb possible. Once the walk reached a hilltop, the procedure getHighestNbr returns NULL and the second phase of the algorithm (presented in Figure 9 as getBestHillExit) is used in order to go down the hill on the opposite side. This algorithm calculates for each neighbor the ratio between the number of its direct neighbors that are not already included in the Trail and the number of those that are already included in the Trail. The algorithm then forwards the message to the neighbor with the highest calculated ratio. This ratio will usually be highest for the neighbor that is the farthest away from the hilltop at the “opposite side” from which the walk climbed the hill. Such a neighbor has the highest probability of leading the walk outside the current hill and towards a yet unvisited hilltop in a neighboring hill. If all of the nodes in the 2-hop neighborhood of the forwarding node are already included in the Trail, we say that the density biased walk has reached a dead end. In such a case we use backtracking until an alternative path is found using either the first or the second phase of the algorithm, or until the walk returns all the way back to the sender of the message. The pseudo-code of the algorithm is presented in Figure 9.

An example of a density biased walk over a network’s virtual topography is presented in Figure 10. The walk starts at the edge of the right hill. The dotted arrows represents steps that have been made using the first phase of the algorithm - reaching a new hilltop using the steepest climb possible; the dashed arrows represents steps that have been made using the second phase of the algorithm - exiting the current hill towards a yet unvisited hilltop in a neighboring hill. The solid arrows represent each node’s highest neighbor.

4.2.2 Data Advertisement

Data Owners use data advertisement in order to advertise their hosted data items at hilltops across the network. Data advertisement is performed by periodically sending advertisement messages along density biased walks. The length of each walk is determined dynamically by the advertisement stopper described in Section 4.2.5. Each advertisement message originates at the advertising data owner and travels throughout the network along a different density biased walk, assuming the network topology changes between advertisements. Each advertisement message contains many tuples of the form (key, data owner, expiration)\(^5\), which we denote by ads. Each ad represents a specific hosted data item and its key is a short string describing that data item (e.g., a URL address, a file name, etc). The ad’s expiration is the time in milliseconds the ad has left to live before it expires. Each hilltop along the walk caches all the ads from the advertisement message along with a timestamp and piggybacks on top of the advertisement message all of the ads located in its own cache before forwarding the advertisement message to the next node on the walk\(^6\). Nodes along the walk that are not hilltops forward the advertising message without caching or changing its content. Each hilltop periodically checks the expiration time and the timestamp of the ads stored in its cache and deletes expired ads. A sorted data structure of ads, sorted based on expiration time, can be used for efficiency. The advertisement frequency and the expiration time of ads are both protocol parameters. In our measurements

\(^5\)The data owner and the expiration are attached to each data item separately in order to support data handoff and piggybacking. Some memory optimizations are possible, but are omitted for brevity.

\(^6\)In case there are too many such items, only part of them are piggybacked on each message.
Upon \texttt{receive(walk\_msg)} do

1. handle\(\texttt{walk\_msg}\); /* according to the message’s type (advertisement, handoff or lookup) */
2. if \(\texttt{walk\_msg.walk\_stopper.isTimeToStop()}\) then return; /* walk stopper stopped the walk */
3. /* forward the message */
4. next\_hop := \texttt{getHeighestNbr(walk\_msg)}; /* first phase */
5. if \(\texttt{next\_hop = NULL}\) then next\_hop := \texttt{getBestHillExit(walk\_msg)}; /* second phase */
6. if \(\texttt{next\_hop = NULL}\) then /* a dead end has been reached */
7. if \(\texttt{Route\_Back.isEmpty()}\) return; /* the backtracking stack is empty - message returned to sender */
8. next\_hop := \texttt{walk\_msg.Route\_Back.pop();} /* backtracking */
9. endif
10. update the Trail, Route\_Back and #Visited\_Hilltops in walk\_msg;
11. send\(\texttt{walk\_msg, next\_hop}\); /* forwarding message */

Upon \texttt{getHeighestNbr(walk\_msg)} do

12. highest\_nbr := NULL;
13. foreach \(q \in (N(1, p) \backslash \{p\})\) do
14. if \((\texttt{highest\_nbr = NULL} \land \texttt{highest\_nbr is higher than q})\) then continue; /* jumping to the next iteration */
15. if \((\texttt{Hilltop}_q \in \texttt{walk\_msg.Trail})\) then continue; /* overcoming most hilltop misconceptions */
16. highest\_nbr := q;
17. endforeach;
18. return \texttt{highest\_nbr};

Upon \texttt{getBestHillExit(walk\_msg)} do

19. best\_exit\_score := 0;
20. best\_exit := NULL;
21. foreach \(q \in (N(1, p) \backslash \{p\})\) do
22. #\texttt{unseen\_neighbors} := \(|N(1, q) \backslash \texttt{walk\_msg.Trail}|\);
23. #\texttt{seen\_neighbors} := \(|N(1, q) \cap \texttt{walk\_msg.Trail}|\);
24. exit\_score := #\texttt{unseen\_neighbors} / #\texttt{seen\_neighbors};
25. if \((\texttt{exit\_score} < = \texttt{best\_exit\_score})\) then continue;
26. best\_exit\_score := exit\_score;
27. best\_exit := q;
28. endforeach;
29. return \texttt{best\_exit};

Figure 9: Density biased walk forwarding algorithm (as preformed by node \(p\))

reported in Section 5 below, we explore the ratio between these numbers. We discovered that setting them
to 15 and 30 minutes respectively gives good results: advertising once in 15 minutes yields low overhead,
whereas an expiration time of twice the advertisement frequency (or more) is enough to ensure high lookup
success ratio, as it gives enough time for the advertisements to spread in the network.

4.2.3 Data Handoff

One of the main invariants of our protocol is that only hilltops maintain data location information. The
fact that our virtual topography contains relatively small number of hilltops per area (Section 3.2) helps
to bound the replication level of the data location information across the network. In order to avoid loosing
the ads contained in the cache of a hilltop while a hilltop switch occurs, we need to preform a data handoff
between the old and the new hilltops. A data handoff is performed on each hilltop switch if and only if the
old hilltop contains some ads in its cache. Once a hilltop steps down and adopts a new hilltop, it forwards
all the ads from its cache, typically ads from different data owners, to its new hilltop and empties its own cache. The ads are wrapped in a handoff message and sent using the same density biased walk described above. The walk is stopped once it reaches the new hilltop as implemented by the handoff stopper described in Section 4.2.5. The new hilltop caches all of the ads from the handoff message and by that keeps them from being lost. The forwarding algorithm, using only its first phase, will forward the handoff message in the first hop to the highest direct neighbor, which will most likely be the new hilltop. Hence, usually after just one hop, the walk is completed. There are some uncommon cases in which more than one hop will be required in order to deliver the handoff message to the new hilltop. However, the first phase of the algorithm delivers the handoff message to the new hilltop greedily by choosing the highest direct neighbor at each hop until reaching the new local maxima. Empirically, as reported in Figure 14(b), the average number of hops required by such a handoff message is about 1.1 hops. When the old hilltop sends the content of its cache to the new hilltop it first has to subtract from the expiration field of each ad the time elapsed since it first received the ad in order to maintain the expiration’s consistency.

The total number of handoff messages sent due to hilltop switches is mainly derived from the average number of hilltops in the network and the mobility properties of the nodes. As a result, the number of such messages does not increase with the total number of nodes in the network (Section 3.2). Moreover, it does not increase with the number of data owners (after the first few), the number of data items hosted by each of them nor with the number of clients in the network. These properties makes the data handoff protocol extremely light weight and scalable.

The proposed protocol treats all nodes in the network as equal. However, one might claim that the nodes currently serving as hilltops have more responsibilities than the rest of the nodes only due to their physical dense location, which may lead to an unequal load balance across the network. This is a concern in
completely static networks, when the network’s topology, and hence virtual topography, does not change at all (e.g., in static sensor networks). However, even minor changes to the network’s topology caused by very limited mobility would result in a constant rotation among hilltops. For example, empirically, in a MANET following the Random Walk mobility model in which nodes’ speed varies randomly between \(0.5 - 2\) m/s, the average time a node serves as a hilltop before stepping down is less than 5 seconds.

4.2.4 Data Lookup

In order to locate data items in the network, a client sends out a query message along the same density biased walk described above. Each query message contains a regular expression defining a set of requested keys. These regular expressions can be exact phrases (e.g., “http://www.cnn.com”) matching only a single key or they can be more general (e.g., *.mp3) matching many possible keys. The query messages are sent along a density biased walk as described above. Whenever the walk reaches a hilltop it searches its cache for the requested key and terminates the walk once such a match is found. The exact condition for stopping the lookup walk is determined by the lookup stopper described in Section 4.2.5. On a successful match, the hilltop at the end of the walk will issue a response message containing all of the matching ads found in its cache. The response message will travel across the reverse path of the query message, back to the client. In case the route back is broken, mostly due to nodes’ mobility, the broken link is bypassed by sending the response message through a common neighbor. Recall that nodes include a set of IDs of their direct neighbors in every heartbeat message.

The average percentage of hilltops that hold in their cache a certain ad, averaged over all of the unexpired ads advertised in the network, is referred to as the knowledge level of the hilltops. For example: if all the hilltops in the network hold every unexpired ad ever advertised, we say that the knowledge level of the hilltops is 1. In order for the 3DLS protocol to achieve high success rates, the knowledge level of the hilltops should be high enough. This is obtained in 3DLS through a self-tuning mechanism as follows: lower/higher knowledge levels of the hilltops will prolong/shorten the average length of a successful lookup walk. In order to help keep the knowledge level of hilltops high enough, the lookup and reply messages piggyback all of the ads cached in the hilltops they pass through along the walk while these hilltops cache all of the ads already piggybacked on top of these messages. The piggybacking mechanism helps to increase the knowledge level of the hilltops in inverse proportion to their current knowledge level because the lower the knowledge level is, the longer the average data lookup and reply walks becomes, resulting in more piggybacking which increases the knowledge level of the hilltops.

As we have discovered empirically (Section 6), piggybacking ads during data lookups and replies is extremely effective at keeping the knowledge level of the hilltops high enough. This is mostly due to the inverse proportion between the knowledge level of the hilltops and the average length of a successful lookup walk. We have discovered that under common network scenarios it is sufficient for the data advertisement protocol to only advertise its hosted ads at a single hilltop (its own) in each data advertisement. These ads will most likely get piggybacked by some lookups and reply walks, and propagate to other hilltops across the network. This can usually be done without compromising the success rate of the protocol. The reason for which we still use longer advertisements is to support wider network scenarios such as when data lookups are scarce or when the network is very large and the data items’ expiration is relatively short (e.g., stock quotes).
Upon Advertisement_Stopper.isItTimeToStop() do
(1) if all of the $\lambda$s seen at hilltops along the walk are empty then return false;
(2) if the weighted average of the $\lambda$s seen at hilltops along the walk $> \frac{1}{2}$ then return false; /* the knowledge level of the hilltops is too low */
(3) return true; /* stopping only at hilltops with a non-empty $\lambda$ measure*/

Upon Handoff_Stopper.isItTimeToStop() do
(4) if $p$ is not a hilltop then return false;
(5) return true; /* stopping at the first hilltop */

Upon Lookup_Stopper.isItTimeToStop() do
(6) if all of the $\lambda$s seen at hilltops along the walk are empty then return false;
(7) if $p$'s cache contains an ad matching the requested key then
(8) update $p$'s $\lambda$ using #Visited_Hilltops;
(9) return true;
(10) endif
(11) if the weighted average of the $\lambda$s seen at hilltops along the walk $\beta > \#Visited_Hilltops + 1$ then return false;
(12) return true; /* stopping only at hilltops */

Figure 11: The walk stoppers algorithm (as preformed by node $p$)

4.2.5 Walk Stoppers

The first intuitive method to determine when a density biased walk of each type (advertisement, handoff and lookup) should be stopped is to assign each of the walk types a predefined length, which is defined as the number of hilltops the walk needs to pass through. Naturally, the length of the handoff walk is set to 1. However, setting the length of the other two types of walks requires making some assumptions regarding the network size. In order for our protocol to be highly scalable and self adaptive, a more sophisticated approach is used.

In our approach, we monitor a measure of the knowledge level of the hilltops. Based on the measure, the advertisement and lookup walk stoppers decide when each walk should be stopped. This measure, denoted by $\lambda$, is defined as the number of hilltops visited along a single successful lookup walk. When a hilltop receives a lookup query on which it can reply, it extracts the number of hilltops visited along the lookup walk including itself ($\lambda$) from the lookup message before it sends a reply message. Each hilltop manages a moving average of the $\lambda$s it extracts. This moving average is denoted by $\tilde{\lambda}$ and is said to be empty if it has not obtained any $\lambda$ values yet. A non-empty $\lambda$ is included in the handoff message whenever a hilltop switch occurs. When a hilltop receives a $\lambda$ measure inside a handoff message it replaces its own $\lambda$ with a weighted average between its previous (possibly empty) $\tilde{\lambda}$ and the $\lambda$ it received inside the handoff message. It is easy to see that the knowledge level of the hilltops $\approx 1/\tilde{\lambda}$ for any $\lambda$ managed by a hilltop in the network. For example, if the average number of hilltops visited along successful lookups is 5, it can be said that on average, each unexpired ad is cached by 1 out of 5 hilltops. Advertisement and lookup stoppers use the $\lambda$s managed by the hilltops visited along the walk in order to determine exactly when a walk should be stopped. The logic behind the advertisement stopper is: The larger the $\lambda$s along the walk are, the lower is the knowledge level of the hilltops in the network. Hence, a longer advertisement walk should be used. The logic behind the lookup stopper is: The larger the $\lambda$s along the walk are, the lower is the knowledge level of the hilltops in the network. Hence, a longer lookup walk should be used before giving up. Recall that due to piggybacking, lookup walks increase the knowledge level of the hilltops in direct proportion to their length. The pseudo-code of the walk stoppers’ algorithm is presented in Figure 11.
The parameters $\alpha$ and $\beta$ used by the walk stoppers are protocol parameters. $\alpha$ is used by the user in order to set the target knowledge level of the hilltops and by that resolving the tradeoff between the efforts spent on advertisement and the efforts spent on lookups. In our experiments, we have set it to be 0.5. The advertisement stopper makes sure the average $\lambda$ in the network will not exceed $\frac{1}{2}$ by responding to higher average $\lambda$s with much longer advertisement walks which, as described in Section 4.2.2, piggyback ads from hilltops along the walk, until the system self stabilizes. $\beta$ is used in order to set the maximum effort spent on lookups of possibly non-existing data items before quitting. We have set it to be 10.

At bootstrap, or if there are no successful lookups in the network, the $\lambda$ measure managed by all of the hilltops in the network is empty. This undesirable state will disable the advertisement and lookup stoppers from ever stopping any advertisement or unsuccessful lookup walks. In such a case the walks will be stopped at the node that initiated the walk after backtracking from the corners of the network as stated in line 7 in the receive procedure (Figure 9). If there are no successful lookups at all in the network, data owners can act as dummy clients searching for their own data items in order to have some non-empty $\lambda$ measures in the network. This situation can be detected by data owners after some of their advertisement walks have backtracked all the way back to them.

### 4.3 The Impact of Mobility

Recall that each hilltop manages a cache of ads and a measure $\lambda$ of its view regarding the knowledge level of the hilltops in the network. At any given time, we can divide the hilltops into two disjoined groups as follows:

1. **Aware Hilltops**: hilltops that currently hold a non-empty $\lambda$ measure.

2. **Unaware Hilltops**: hilltops that currently hold an empty $\lambda$ measure.

Each successful data lookup provides another $\lambda$ measure to one of the hilltops. If a data lookup is resolved by an unaware hilltop it immediately becomes an aware hilltop. In static networks, where the hilltops are constant, the number of unaware hilltops will never increase and potentially decrease with each successful data lookup, typically resulting in an overwhelming majority of aware hilltops. In mobile networks, where topographical changes are frequent, the number of unaware hilltops will increase whenever a hill splits into two separate hills. Each hill split always results in a new and unaware hilltop. Frequent hill splits balance the decrease in unaware hilltops as a result of successful data lookups. As a result of these balancing phenomena, in mobile networks there is typically a slight majority of unaware hilltops. Hill splits would also cause the knowledge level of the hilltops to constantly decrease in mobile networks unless balanced by relatively long advertisement walks. Whereas in static networks the knowledge level of the hilltops would decrease much slower (only due to ads’ expiration) unless balanced by much shorter advertisement walks.

By examining the algorithm of the advertisement stopper (Figure 11) we see that the advertisement stopper can only stop an advertisements walk at aware hilltops. In mobile networks, where there is a slight majority of unaware hilltops, advertisement walks are typically longer than required in order to keep the knowledge level of the hilltops constant. This is because most of the hilltops along the advertisement walk are unaware hilltops that cannot stop the walk. The longer advertisement walks increase the knowledge level of the hilltops, which in turn results in shorter lookup walks in these networks. In static networks, where there is an overwhelming majority of aware hilltops, advertisement walks are the shortest that is needed in order to keep the knowledge level of the hilltops constant. As a result, the lookup walks are a bit longer in static networks.
<table>
<thead>
<tr>
<th>DLS Parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data owners count</td>
<td>5% of the nodes</td>
</tr>
<tr>
<td>Data items per data owner</td>
<td>1</td>
</tr>
<tr>
<td>Data owners per data item</td>
<td>1 (each data owner has a unique data item)</td>
</tr>
<tr>
<td>Clients count</td>
<td>100% of the nodes</td>
</tr>
<tr>
<td>Average lookup frequency per client</td>
<td>1 per 6 simulated hours</td>
</tr>
</tbody>
</table>

Figure 12: The default DLS usage used in the simulations

4.4 Range Queries

In a range query, the lookup walk may be directed to stop at the first hilltop in which some matching is found, similarly to a single item query. In this case, the reply returns all matchings found in that hilltop. Alternatively, for a more complete answer, a range query can continue until the walk passes through a certain number of hilltops, until it finds a certain number of matchings or any combination of the two. The number of required hilltops/matchings can be, for example, a function of the knowledge level in the system. Exploring this issue further is left for future work.

5 Simulations

In this section, we evaluate the performance of our protocol in a simulated environment. We conducted extensive simulations using the JIST/SWANS simulator to evaluate the performance of our protocol under different network characteristics and using different DLS configurations (Section 5.1). In Section 5.2 we compare the performance of DLS with the performance of flooding based protocols. In Section 5.3 we compare the performance of DLS with the performance of the GCLP protocol [29]. GCLP is a classic protocol that uses geographical knowledge in order to provide data location services while DLS does not. The comparison between GCLP and DLS shows that DLS is much more robust, scalable and light weight than GCLP in spite of the obvious advantage GCLP has by using geographical knowledge. In Section 5.4 we present an analytical comparison between DLS and DHTs like Pastry [26] and Chord [28].

Setup: The default network characteristics we used in all of our simulations are presented in Figure 6. The default data location service usage parameters and the DLS configuration we used in the simulations are presented in Figure 12 and Figure 13 respectively. Most of the parameters are checked individually for their sensitivity by changing each of them to a range of values while the rest of the parameters are set to their default values. Each simulation lasted for one simulated hour and started after a 30 minutes initialization period, which was enough for the ads to be well propagated across the network. Each data point was generated as an average of 10 runs.

Metrics: During each simulation we tested the following measures:

- Success Rate: the percentage of data lookup queries that has been answered correctly by a reply message that has been received by the client.
- Adv length (#Hops): The average number of hops traveled by each advertisement walk.
<table>
<thead>
<tr>
<th>3DLS Parameters</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heartbeat cycle time</td>
<td>3 simulated seconds</td>
</tr>
<tr>
<td>Advertisement cycle time per data owner</td>
<td>15 simulated minutes</td>
</tr>
<tr>
<td>Ads’ expiration time</td>
<td>30 simulated minutes</td>
</tr>
<tr>
<td>Advertisement stopper const ( (\alpha) )</td>
<td>0.5 (target knowledge level of the hilltops)</td>
</tr>
<tr>
<td>Lookup stopper const ( (\beta) )</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 13: The default 3DLS configuration used in the simulations

- Adv length (#Hilltops): The average number of hilltops traveled by each advertisement walk.
- Lookup length (#Hops): The average number of hops traveled by each lookup walk.
- Lookup length (#Hilltops): The average number of hilltops traveled by each lookup walk.
- Handoff length (#Hops): The average number of hops traveled by each handoff walk.
- Total Messages Sent: the total number of messages passed by the tested DLS protocol to the MAC layer during the simulation. This number includes all types of messages except for the heartbeat messages which are analyzed separately.
- Average Message Size: the average size in bytes of the messages passed by the DLS protocol to the MAC layer. This number includes all types of messages except for the heartbeat messages.
- Replication Level: The average percentage of nodes which hold in their cache an ad of a certain data item, averaged over all of the data items in the network.
- #Hilltops: The average number of concurrent hilltops in the network throughout the simulation.
- Hilltop Coverage: The simulation area divided by #Hilltops. For simplicity, we present the result in transmission disk units, i.e., we divide it by \( \pi r^2 \) where \( r \) is the transmission range.
- Hilltop Switch Rate: The average number of hilltop switches per simulated second.

5.1 3DLS Performance

We first evaluate the performance of our protocol under different network characteristics and using different 3DLS configurations. The evaluation results are presented in Figure 14. Figures 14(a) and 14(b) show 3DLS’s scalability. The number of nodes in the tested network was changed together with the test area in order to maintain a constant node density of 10 neighbors per node. Figure 14(b) shows that the number of hilltops in an average advertisement walk increases with scale. This is a result of the need to support the same hilltop knowledge level \((0.5)\) in a much larger network. Interestingly, Figure 14(a) shows that the number of hops in such a walk slightly decreases with scale. This is a direct result of the edges effect in small scale networks. In small scale networks a dead end in the density biased walk forwarding algorithm (Figure 9) is reached much more often resulting in frequent backtracking. In Figure 14(b) we see that the number of hilltops in an average lookup walk is about 2. This is a direct result of setting the hilltop knowledge level \((\alpha)\) to 0.5. We also learn from Figures 14(a) and 14(b) that the number of hilltops in the network is linear to the ratio between the test area and the area covered by one transmission disk. The linear
Figure 14: 3DLS’s sensitivity to various parameters
coefficient is about 3.2 which means that the average area covered by a hill is equivalent to 3.2 transmission disks.

Figure 14(c) exhibits the performance of 3DLS in various network densities. The density has been changed by varying the number of nodes placed on the same 1400\times 1400 m^2 test area. The X-axis reflects the average number of direct neighbors each node has. Networks where nodes have an average of 5 direct neighbors are not connected with high probability (Gupta & Kumar [16]), which explains the longer lookup walks and the lower success rate in these settings. Longer lookup walks are the result of lower knowledge levels at the hilltops, which also triggers longer advertisement walks. Once the network becomes connected (around 7.5 neighbors in average), the success rates reaches above 99\% and the walks become much shorter. Interestingly, even in poorly connected networks, the longer advertisement walks in these settings yield success rates close to 90\%, which demonstrates the self-adaptivity of 3DLS. We can also learn from Figure 14(c) that the number of hilltops is indeed independent of the total number of nodes in well connected networks.

Figure 14(d) shows the adaptivity of 3DLS to nodes’ lookup frequency. Because we piggyback ads on top of each lookup and reply message, less frequent lookups decreases the knowledge level of the hilltops prolonging each lookup walk. This immediately causes the advertisement walks to become much longer in order to keep the knowledge level of the hilltops constant even if lookups are very scarce. In Figures 14(e) and 14(f) we see the embedded adaptivity mechanism of 3DLS compensating for two more network scenarios with longer advertising walks. These network scenarios would have otherwise decreased the knowledge level of the hilltops together with the success rate of the protocol. In Figure 14(e) we changed the number of data owners in the network. Recall that each additional data owner helps to propagate ads advertised by the rest of the data owners by piggybacking them on its own advertisement messages. In Figure 14(e) we see that when the number of data owners is low, each advertisement walk becomes longer in order to compensate for the lack of help received from other data owners. In Figure 14(f) we changed the ads’ expiration time. We see that shorter expiration time is again compensated with longer advertisement walks in order to keep the knowledge level of the hilltops constant and by that keeping the success rate above 99\%.

Figure 14(g) exhibits the performance of 3DLS in various network speeds. We can see that in faster networks the hilltop switch rate increases significantly. Frequent hilltop switches trigger frequent hill splits and mergers. Whenever a hill splits, a new hilltop with empty cache emerges. Whenever two hills merge, two hilltops with potentially non empty caches unite. Frequent hill splits and mergers constantly decrease the knowledge level of the hilltops. As a result, in fast moving networks, the advertisement walks become much longer in order to keep the knowledge level of the hilltops constant in face of frequent hill splits and mergers. We also see that in high speed networks the success rate drops to about 85\%. This is because reply messages along the reverse path are more likely to get lost in high speed networks.

Figure 14(h) presents the performance of 3DLS using different heartbeat frequencies. We can see that a lower heartbeat frequency results in longer advertisement/lookup walks. This is because nodes’ densities change faster than their virtual heights since the virtual height of each node is only recalculated immediately before sending a heartbeat message. This means that the calculated height that each node uses is somewhat stale when it is being used by the walks, resulting in slightly longer walks. We also see that the hilltop switch rate decreases when using lower heartbeat rates. This increases the knowledge level of the hilltops due to the fewer hilltop splits and mergers. However, when using lower heartbeat rates, the average distance between switching hilltops during a hilltop switch increases. If this distance exceeds the transmission range, the new hilltop can hold the hilltop position for some time before the old hilltop notices a higher neighbor and initiates a handoff message. In such cases, which are more common as the heartbeat rate decreases, new hilltops hold an empty cache for longer periods of time. This increases the average number of hilltops in
each lookup walk, which cancels out the effect of the increased knowledge level of the hilltops.

In Figure 14(i) we show the performance of 3DLS in different mobility models. In addition to the Random Walk model (denoted by RW), we tested the Random Waypoint model (denoted by RWP) with pause time set to 0, and static networks with random placement. As described in Section 4.3 in static networks there is an overwhelming majority of aware hilltops supporting very short advertisement walks, mostly passing through one or two hilltops. The advertisement walks are deterministic and will follow the exact same route whenever initiated in static networks. This reduces the uniformity of the ads’ dissemination resulting in a lower success rate in static networks. In the Random Waypoint model the nodes’ density in the center of the network is significantly higher than the nodes’ density in the edges of the network. This property is significant in our density-driven protocol. The Random Waypoint model causes the resulted virtual topography to be less uniform throughout the network. A typical topography as a result of the Random Waypoint model would be a high and wide ridge in the center of the topography, which includes only a few hilltops. The resulting topography also exhibits a gradual decrease in nodes’ height (with hardly any hilltops) towards the edges of the network. There are significantly fewer hilltops in such a topography than in a uniform topography, which means that the walks must pass through more hops in order to visit the same number of hilltops on their way. The hilltops at the center of the network have a very high knowledge level (usually complete) and most of the advertisement and lookup walks are directed towards the center of the network. The slightly lower success rate in the Random Waypoint model is due to the poor connectivity at the edges of the network.

In addition, we tested 3DLS’s sensitivity to the number of distinct data items per data owner. We have increased the number of data items per data owner up to a 1000 (i.e., 50,000 data items in the network). There was no significant change in the success rate or in the number of messages sent throughout the simulation. However, the average packet size has increased in linear proportion to the total number of data items in the network. Yet, this is also controllable: it is possible to optimize the protocol such that only a fraction of the ads are piggybacked on each message with various heuristics on how to chose them. Such an optimization will automatically prolong the advertisement walks in order to keep the knowledge level of the hilltops constant.

5.2 3DLS vs. Flooding-Based Protocols

In this section, we compare the 3DLS protocol with the following flooding-based protocols which we implemented using the JIST/SWANS simulator:

1. Advertisement Broadcast (denoted by ABC): On each data advertisement the data owner wraps the ads representing its hosted data items inside a single message and broadcasts this message throughout the network by flooding the network. Each node caches the ads it receives from each broadcast and removes ads from its cache once they expire. On each data lookup the client resolves the request locally by searching its own cache.

2. Lookup Broadcast (denoted by LBC): Data owners do not advertise their hosted data items. On each data lookup the client floods the network with a data lookup query. Once a data owner receives a data lookup query requesting one of its hosted data items, it sends a response message along the reverse path, back to the client. In case the route back is broken, mostly due to nodes’ mobility or failure, a contained 2-hop broadcast takes place in order to locate the next node on the route back skipping over the unreachable node.

When simulating the protocols ABC and LBC, the same network characteristics (presented in Figure 6) and DLS usage (presented in Figure 12) have been used. In ABC, the advertisement cycle time and ads’
expiration time were set to 15 and 30 minutes respectively, just like in 3DLS. Notice that in LBC data owners do not advertise their hosted data items at all. No protocol other than the tested flooding-based protocol was active in the network during the simulations (including any heartbeat protocol).

Figure 15 exhibits the scalability of the 3DLS protocol versus the scalability of the flooding-based protocols. The number of nodes in the tested network was changed together with the test area in order to maintain a constant node density of 10 neighbors per node. In Figure 15(a) we show that the success rate of LBC drops in large scale networks faster than in ABC and 3DLS. This is because the average route length from the client to the data owner that holds the requested data item increases significantly in large scale networks. This increase in the average route length increases the probability for every reply message to get lost on the way, mostly due to nodes’ mobility. In 3DLS, the average length of a reply message is very short due to the high knowledge level of the hilltops (which was set to be 0.5) and the uniformity of the hilltop distribution across the network. In ABC the queries are resolved locally and hence the highest success rate.

Figure 15(b) shows that the total number of messages sent by ABC and by LBC grows significantly in large scale networks. In comparison, the increase in the total number of messages sent by 3DLS is extremely mild. This is due to the high cost of broadcast messages in large scale networks. Recall that 3DLS does not use broadcast at all. As stated in the metrics definition in Section 5, the total number of messages sent by 3DLS does not include heartbeat messages. This is for two reasons: The overhead of heartbeats is constant for a given network size and heartbeat frequency, and does not depend on the usage pattern of the service. Moreover, there are many protocols with different purposes (e.g., routing, broadcasting, etc.) in which nodes need to know their direct neighbors. Many of these protocols utilize the same heartbeat mechanism for this. Hence, the cost of the heartbeat protocol is shared among all of these protocols, and should not be attributed to only one of them.

Figure 15(c) shows that the average size of each message sent in the network is significantly higher in 3DLS than in the flooding-based protocols. This is a direct result of piggybacking ads on top of every non-heartbeat message sent by 3DLS. We also show that the average message size in 3DLS is linear to the number of data items in the network. Recall that the number of data owners is 5% of the number of nodes in the network, hence, the number of data items in the network is linear to the network scale.

Figure 16 compares the sensitivity to nodes’ density of the 3DLS protocol versus the flooding-based protocols. The density has been changed by varying the number of nodes placed on the same 1400X1400 m² test area. The X-axis reflects the average number of direct neighbors each node has. In Figure 16(a), we
show that the success rate in LBC drops in weakly connected networks faster than in 3DLS and much faster than in ABC. This is because in weakly connected networks a client does not always have a path to the data owner hosting the requested data item when the client initiates the lookup. In such a case the lookup will not succeed in LBC. In ABC and 3DLS such lookups can still succeed because data advertisements disseminate the data location information to other nodes in the network. Notice that the client will still fail to retrieve the requested data item from the data owner if there is no path between them. The results presented in Figures 16(b) and 16(c) are a direct result of the increase in the total number of nodes, and hence, in the number of data owners and data items in denser networks.

Figure 17 compares between 3DLS’ sensitivity to nodes’ speed and the sensitivity of the flooding-based protocols. In Figure 17(a), we show that the success rate of LBC drops with each increase in nodes’ speed. This is because a reply message has a higher probability of getting lost along the route back when the nodes’ speed is higher. We also show that nodes’ speed hardly effects the success rate of ABC. The sensitivity of 3DLS to nodes’ speed has been described in Section 5.1. In Figure 17(b), we show that nodes’ speed does not effect the total number of messages sent in the flooding-based protocols. Differently, in 3DLS, increasing the nodes’ speed causes an increase in the average length of advertisement and lookup walks and the frequency of hilltop switches (see also Figure 14(g)). In Figure 17(c), we show that nodes’ speed does not effect the average message size in flooding-based protocols. However, in 3DLS, increasing the nodes’ speed causes an increase in the average length of advertisement walks, which piggybacks ads as it goes.
This results in an increase in the average message size. Notice that in static networks, there are no hilltop switches in 3DLS, and hence, no handoff messages at all. This causes the average message size in static networks to be slightly higher than when mobility is introduced.

5.3 3DLS vs. GCLP

In this section, we compare between the performance of 3DLS and the performance of GCLP [29]. GCLP is a geography-based column-row quorum-type DLS protocol [12]. In GCLP, data owners advertise their hosted data items and clients propagate their lookup queries in cross-shaped trajectories, thus guaranteeing two intersections. Lookup queries are answered by nodes at the intersection of the advertisement and query trajectories. GCLP incorporates additional improvement: a node may pick whether to forward an advertisement message or not based on the proximity of the data owner. In case of multiple advertisements for the same data items, only the closest copy is advertised further on. This results in a dynamic update grid as presented in Figure 18.

Comparing between 3DLS and GCLP is not straightforward. In order for the column-row quorums to intersect in mobile networks GCLP uses a relatively short advertisement cycle time (about 30 seconds). 3DLS does not have this problem and so it uses a relatively long advertisement cycle time (15 minutes). Due to its short advertisement cycle time and its geographical knowledge, GCLP also provides its clients the physical location of the data owners hosting the requested data items. This is possible by adding the data owners’ geographic location to each advertisement message. This information can help the client to contact any of the provided data owners in order to retrieve the requested data item. In 3DLS, the lack of geographical knowledge and the long advertisement cycle time prevents the protocol from providing the geographic location of the appropriate data owners. When comparing 3DLS and GCLP one has to take into account the additional geographic information GCLP provides to its clients, which 3DLS does not.

7Octopus is also based on similar concepts [21].
In order to fairly compare the costs of these protocols, we need to change one of them so that they provide the same service to their clients. GCLP requires frequent advertisements in order to guarantee the column-row quorums’ intersection. The ability to provide the appropriate data owners’ geographic information is a positive side effect of the frequent advertisements and it incurs minimal overhead to the messages’ size. Therefore, modifying GCLP so it will only provide the appropriate data owners’ ID will hardly change the traffic induced by the protocol and hence, would be unfair. Enhancing 3DLS to provide the data owners’ geographic location in addition to their IDs requires 3DLS to obtain geographical knowledge and to send advertisements much more frequently. This would significantly impact the traffic induced by the protocol. Having to choose one of the suggested changes, we think it is more fair to change the advertisement frequency of 3DLS than to modify GCLP.

In the Augmented 3DLS protocol, denoted A-3DLS, nodes have geographical knowledge and they include their physical location inside every advertised ad. It is important to point out that the A-3DLS protocol does not utilize the geographical knowledge in managing its advertisements and lookups; the geographical location of a data owner is simply included in its ads and therefor also returned as part of a lookup reply. A-3DLS, just like GCLP, uses a relatively short advertisement cycle time of 30 seconds and a short ads’ expiration time of 90 seconds. This is required in order to provide relevant location information in face of nodes’ mobility. Note that A-3DLS is much more costly than 3DLS: its advertisements are 30 times more frequent and each ad is 33% larger (16 bytes instead of 12 bytes). We have compared between GCLP, A-3DLS and 3DLS.

From the results presented in [29], it is clear that the success rate of GCLP is highly sensitive to the number of data owners per data item in the network (this parameter is referred in [29] as the number of servers in the network). As reported in [29], if there is only one data owner per data item in the network, the success rate of GCLP is extremely low. Hence, GCLP is most suitable when there are few data owners per data item in the network. In order to test GCLP in its suitable environment, we have changed the default number of data owners per data item to be 10 throughout this comparison. This means that each data item in the network is hosted by exactly 10 data owners. Each data owner still hosts only one data item (which is not unique). The rest of the default parameters are exactly as presented in Figures 6, 12 and 13.

Nodes in GCLP require knowledge regarding their direct neighbors in order to forward the advertisement and lookup messages along a cross-shaped trajectories. In order to obtain such knowledge, GCLP uses a heartbeat protocol much like 3DLS (and A-3DLS). In our simulations, GCLP, 3DLS and A-3DLS use the same heartbeat frequency and hence generate exactly the same number of heartbeat messages throughout each simulation. There is only a negligible difference in the average size of the heartbeat messages in these protocols. In the comparison between these protocols we discard the cost of the heartbeat protocol since it is equal in all of the compared protocols.

Figure 19 exhibits the scalability of 3DLS, A-3DLS and GCLP. The number of nodes in the tested network was changed together with the test area in order to maintain a constant node density of 10 neighbors per node. In Figure 19(a), we see that the success rate of GCLP drops fast in large scale networks. This is due to the increased difficulty of constructing the update grid when the average distance between data owners that host the same data item increases in large scale networks. In Figure 19(b), we see that the total number of messages sent in GCLP is higher than in A-3DLS. This is due to the longer advertisements routes and much longer lookup routes in GCLP compared to A-3DLS. The total number of messages sent by 3DLS is much smaller due to the low advertisement frequency. In Figure 19(c), we see that the average message size in 3DLS and A-3DLS is linear to the number of data items in the network while in GCLP it is constant. Notice that each ad is 33% larger in A-3DLS in comparison to 3DLS. However, in 3DLS, longer advertisement walks are required and the average number of ads piggybacked on each message is larger.
However, even in large scale networks, the average message size is still much smaller than the standard 802.11 MTU. In addition, recall that this size can be capped by setting the maximum number of items piggybacked on each message to a constant. In Figure 19(d), we see that the average replication level in GCLP is much higher than in 3DLS and A-3DLS. This is because in GCLP every node that is located along the cross-shaped trajectory of the advertisement message caches the ad while in 3DLS and A-3DLS only hilltops cache the ads. In large scale networks the replication level of GCLP drops because the ratio between the number of nodes along the cross-shaped trajectory and the total number of nodes in the network drops.

Figure 20 shows the performance of 3DLS, A-3DLS and GCLP in various network densities. The density has been changed by varying the number of nodes placed on the same 1400x1400 m$^2$ test area. The X-axis reflects the average number of direct neighbors each node has. In Figure 20(a), we see that the success rate of GCLP drops fast in sparse networks. This is due to the increased difficulty of constructing the update grid when the average number of direct neighbors drops in sparse networks. A-3DLS has a higher success rate than 3DLS in sparse networks due to the fact that advertisements in A-3DLS are much more frequent, meaning that it reacts much more quickly to situations where the knowledge level of the hilltops is lower than specified (recall that $\alpha = 0.5$). In Figure 20(b), we see that the total number of messages sent in GCLP is higher than in A-3DLS in strongly connected networks. In sparse networks the total number of messages sent by GCLP is lower than in A-3DLS. This is due to the fact that its advertisements and lookup routes fail to reach their destination, as is also evident in the low success rate of GCLP in Figure 20(a).
In 3DLS and A-3DLS, while there are more data owners in denser networks, each advertisement walk is shorter. The total number of messages in 3DLS is much smaller due to the long advertisement cycle time. In Figure 20(c), we see that the average message size in 3DLS and A-3DLS is growing with the number of data items in the network while in GCLP it is constant. In Figure 20(d), we see that the average replication level in GCLP increases with density in sparse networks and decreases with density in dense networks. This is because cross-shaped trajectories abort much sooner in sparse networks leading to lower replication level. In very dense networks the replication level of GCLP drops because the number of nodes in the network increases while the number of nodes on the cross-shaped trajectories stays the same. As mentioned before, due to the frequency of advertisements, A-3DLS reacts more quickly to drops in the knowledge level, which is why its replication level is a bit higher than that of 3DLS.

Figure 21 shows the performance of 3DLS, A-3DLS and GCLP in various network speeds. In Figure 21(a), we see that the success rate of GCLP increases with speed in low speed networks and decreases with speed in high speed networks. This is because when nodes hardly move, the cross-shaped trajectories tend to visit the same nodes. This results in a very low replication level increasing the probability that a lookup query will reach a void in the network before finding the requested data. On the other hand, as the mobility increases, so does the number of nodes hosting each unexpired ad. Consequently, data is replicated to a larger number of nodes and lookups have a higher chance of finding a node that stores the data before aborting. In high speed networks, lookups tend to abort due to poor direct neighbors awareness. The
Figure 21: The performance of 3DLS, A-3DLS and GCLP in various network speeds

sensitivity of 3DLS to nodes’ speed has been described in Section 5.1. We see that A-3DLS has a higher success rate than 3DLS in high speed networks. This is simply because it reacts faster to situations where the knowledge level of the hilltops is lower than specified. In Figure 21(b), we see that the total number of messages in 3DLS and in A-3DLS increases with speed unlike in GCLP. GCLP is not a self tuning protocol and does not respond to higher speeds with longer advertisement walks as 3DLS and A-3DLS do. In Figure 21(d), we see that the average replication level in GCLP increases with speed in low speed networks and decreases with speed in high speed networks. Similarly to what happens to the success ratio, this is because in low speed networks, repeated cross-shaped trajectories tend to include exactly the same set of nodes. At the other end, in high speed networks, some cross-shaped trajectories abort due to poor direct neighbors awareness among the nodes, which results in a drop in the replication level.

Figure 22 exhibits the sensitivity of 3DLS, A-3DLS and GCLP to the number of data owners per data item in the network. In Figure 22(a), we see that the success rate of GCLP drops fast as data items become rare. This is due to the increased difficulty of constructing the update grid when the average length of each cross-shaped trajectory increases. The success rate of A-3DLS is a bit lower than in 3DLS when there is only one data owner per data item in the network. This is because of situations in which a data owner advertises its data item to only one hilltop with a very low \( \lambda \) measure (e.g., 1). In such cases, it is likely that other advertisement walks reaching the same hilltop will be stopped immediately at this hilltop and will not propagate this ad further. On the other hand, in 3DLS there is a higher probability that lookup walks
Figure 22: The sensitivity of 3DLS, A-3DLS and GCLP to the number of data owners per data item in the network.

will piggyback this ad or increase the $\lambda$ measure of this hilltop before the ad expires. In Figure 22(b), we see that the total number of messages sent by GCLP decreases when more data owners host the same data item. This is due to the the improved forwarding algorithm introduced in GCLP, allowing nodes to only forward advertisements from the closest data owner. As a result, some cross-shaped trajectories are aborted when there are more data owners advertising the same data item. The total number of messages sent in 3DLS and A-3DLS reduces when there are multiple data owners for the same data item. This is because each data owner advertises its own data item, and hence the same data item gets multiple advertisements in each cycle. The result is an increase in the knowledge level, which yields shorter walks. In Figure 22(c), we see that the average message size in 3DLS and A-3DLS decreases as data items become abundant. This is a direct result of the piggybacking optimization used when data items are not unique (Section 4.1). In Figure 22(d), we see that the average replication level in all of the tested protocols decreases as the number of data owner increase. This is because each of the protocols uses its own optimization (described above) designed to reduce ads’ propagation in face of data items’ duplication.

5.4 3DLS vs. DHTs

In this section we compare between 3DLS and DHTs. A DHT is a distributed data structure that provides hash-table-like semantics. A DHT is a common approach for implementing a Data Location Service over
the Internet. There have been few attempts to implement a DHT over MANETs (Ekta [24], MADPastry [32]
to name a few). A DHT associates every data item in the network to a single node, which is in charge of
resolving queries regarding that data item. The main challenge of any DHT is locating the node that is in
charge of a specific data item, given the data item’s key. Each DHT-based DLS uses a different protocol
in order to forward its advertisements/lookup messages to the node that is currently in charge of the corre-
sponding data item. The efficiency of a DHT trades off the rigidity of the structure it maintains, the size of its
routing tables, also known as finger tables, and the number of routing steps it needs to reach the target node.
In particular, most DHTs use multiple multi-hop routing steps on each data advertisement/lookup. These
routes are typically much longer than the optimal route between the source and the target node. Some DHTs
reduce the number of multi-hop routing steps by increasing the finger table managed by each node. Other
DHTs reduce the length of the average multi-hop routing step along advertisements/lookups by introducing
some locality considerations.

Due to the plethora of DHTs, it is hard to find a representative implementation to compare against.
Instead, we are comparing 3DLS to a theoretical Perfect DHT denoted by PDHT. We assume that all of
the nodes in this perfect DHT (PDHT) know for any given data item, which node in the network is currently in
charge of that data item. We assume that this knowledge is obtained using an internal oracle and without any
communication. When a node using PDHT initiates a data advertisement/lookup, it only has to establish a
route between itself and the known target node, and send its message along that route. We are assuming that
PDHT is using some underlying routing protocol, which, given a target node, establishes the shortest route
to that node. PDHT is clearly at least as efficient as any other known DHT.

Notice that in PDHT we ignore the issue of replication. However, replication is typically done in DHTs
at the successors of the target nodes, since they are used for fault-tolerance rather than to expedite lookups.
Moreover, our comparison below is with the asymptotic behavior of PDHT. Hence, even replication at
additional fixed number of nodes (other than the successors of the target) would only reduce the lookup
cost of PDHT by a small constant. If PDHT uses a higher replication level, it can significantly increase the
memory usage of the protocol. The replication level of 3DLS is further discussed in Section 7.

In PDHT, the average cost of each data advertisement, data lookup and data location reply is the shortest
route length between the source node and the target node, plus the cost of establishing such a route. We
assume that nodes are distributed uniformly across the network and that the network is not static. Under
such assumptions, the average length of an advertisement/lookup/reply route equals to the average length of
the optimal route between two randomly chosen nodes in the network. According to Gupta & Kumar [16],
the average length of an optimal route between two randomly chosen nodes in a connected network of
$n$ nodes is $O\left(\sqrt{\frac{n}{\log n}}\right)$. So, we approximate the average length of an advertisement/lookup/reply route in
PDHT by $\sqrt{\frac{n}{\log n}}$. The cost of establishing such a route is hard to estimate because it strongly depends on
the state of the nodes’ routing tables at that time, which depends on the nodes’ previous communications.
We know that the real cost of establishing such a route is somewhere between $0$ messages (assuming perfect
topological knowledge) and $n + \sqrt{\frac{n}{\log n}}$ messages (assuming zero topological knowledge). We assume that
when nodes have zero topological knowledge and they want to establish a route to some target node, they
broadcast a route request across the network and receive a route reply from the target node itself$^3$.

Figure 23 compares the average cost of an advertisement/lookup/reply message between 3DLS and
PDHT. We show the cost of these messages as a function of the network scale. The data for the 3DLS pro-

$^3$Notice that $\sqrt{\frac{n}{\log n}}$ is also a lower bound on a perfect geographic DHT in which in every step of the DHT the message is
forwarded to a nearby node without the use of routing.
Figure 23 shows that the cost of lookup and reply messages in 3DLS is lower than their lower bound in PDHT. We also see that the cost of advertisement messages in 3DLS is slightly higher than the corresponding lower bound for PDHT for small and medium networks, but much lower than their higher bound. Moreover, for large networks (above 2,000 nodes), even the cost of advertisements is lower than the lower bound on PDHT. In addition, most DHT implementations do not support range queries while 3DLS does. Recall that the average message size in 3DLS is linear to the number of data items in the network while in PDHT it is constant.

We now discuss the maintenance costs of 3DLS and PDHT. The only maintenance 3DLS requires is the data handoff protocol (Section 4.2.3) and the heartbeat protocol. The maintenance of PDHT includes handling continual node arrivals, departures, and network partitions, as well as detecting them through heartbeats. Such events typically incur some communication in order to reassociate some of the advertised data items to different nodes in the network. When assuming that nodes in PDHT have perfect topological knowledge and therefore can locally calculate a route to any node, we must also assume that this knowledge incurs substantial maintenance in mobile networks.
6  Related Work

A substantial amount of research has been devoted to data location services. Most research has been directed towards large scale P2P (peer-to-peer) networks. However, data location services in mobile ad-hoc networks is attracting growing attention in recent years. In this section we give a brief overview of related work on location services in mobile ad-hoc networks.

6.1  Conventional Distributed Hash Tables

Distributed Hash Tables (DHTs) such as Chord [28], Pastry [26], Tapestry [33] and CAN [25] provide very efficient key-based overlay routing. Hence, when trying to build a location service, one could simply employ the conventional DHTs that have been successfully used in the Internet. However, conventional DHTs are ill-suited for a naïve deployment on top of MANETs. This is due to the follow reasons:

1. In conventional DHTs, two overlay neighbors are not likely to also be physical neighbors. Therefore, overlay hops often incur unnecessarily long physical routes, which can cause a lookup query to literally zigzag through the physical network. While mechanisms have been proposed to alleviate this problem (namely in Pastry [26] and Tapestry [33]), this still poses a severe problem in MANETs.

2. Due to nodes’ mobility, routes in MANETs are usually quite volatile and break quickly. Therefore, the communication overhead of a DHT quickly increases when the physical route to carry out an overlay hop has to be (frequently re-) established by the underlying routing protocol. For example, it is easy to imagine a situation where a lookup requires two overlay hops, both of which have to have their physical routes discovered through broadcasting by the routing protocol. Clearly, in that case, one would have been better off broadcasting the key lookup itself in the first place instead of triggering two broadcasts.

3. In order to guarantee routing convergence and consistency, DHTs have to periodically maintain their overlay routing tables. Depending on the size and structure of a DHT’s routing table, the maintenance can generate a significant amount of traffic, which can be a problem given the limited bandwidth in MANETs.

6.2  Unstructured Data Location Services

An unstructured version of a DHT can be found in LMS [22]. LMS enables nodes to publish data items and to retrieve data items published by others. Each data item published has a key in the same ID space as the nodes. The protocol then attempts to store every published data item at nodes that are close to its key in the (circular) ID space. In terms of the distance between the node and the key, they do not find the global minimum (as in a conventional DHT), but rather a local minimum. Publishing a data item involves storing replicas of the data item at a number of randomly selected local minima; retrieving a data item involves querying randomly selected local minima until one is found that holds a replica. Intuitively, the more replicas are placed, the easier it will be to find one of them. The process of finding multiple local minima is accomplished by performing a random walk through the overlay before actively looking for a local minimum.

Gnutella [2] is one of the most-studied unstructured peer-to-peer network. GIA [6] improves on Gnutella using topology adaptations, flow control and biased random walks. GIA dynamically adapts the networks topology in order to create and maintain some well connected “supernodes” (which function as rendezvous
points) while routing all data advertisement/lookup messages to those “supernodes”. Notice that in Internet
P2P systems, the network’s logical topology is controllable, whereas in MANETs, the physical topology
cannot be controlled. Hence, maintaining a logical topology that differs from the physical one requires
expensive routing.

A probabilistic quorums approach to location services has been studied in [13]. In particular, several
approaches to implementing probabilistic quorums have been investigated and analyzed both theoretically
and by simulations. All the approaches in [13] require a replication level of \( O(\sqrt{n}) \) and either the adver-
tisement or lookup (or both) either require routing, or limited range broadcast, or random walks of length
\( O(n) \). 3DLS is more efficient, but has no analytical probabilistic proof.

### 6.3 Distributed Hash Tables Designed for MANETs

Recently, there have been several attempts, e.g., Ekta [24], KELOP [4], ISPRP [8], MADPastry [32] to
efficiently implement DHTs in MANETs.

In Ekta [24], two opposite options for designing a DHT in MANETs are explored. In the first design, a
DHT is directly layered on top of a multi-hop routing protocol, with minimum modifications to the routing
protocol. This approach is thus similar to implementing a DHT in the Internet. This approach, while
consistent with the layered principle of the ISO model of networking, makes it difficult to exploit many
optimization opportunities from the interactions between the DHT protocol and the underlying multi-hop
routing protocol. Ekta adopts the opposite approach, that is, to fully integrate the functions performed by
the DHT protocol and by the routing protocol. With this integration, the routing data structures of both the
DHT and of the routing protocol, e.g., the route cache of DSR, are integrated into a single structure. This
can maximally exploit the interactions between the two protocols to optimize the routing performance.

In KELOP [4], each node has a hash key in the same space as the data items’ keys. As usual, the node
whose own hash key is numerically closest to a given key among all participating nodes is responsible for
that key. The main concept of KELOP is that it does not maintain any explicit structure. Instead, it solely
relies on the local route cache of the given routing protocol. When a data item is advertised or requested, a
message containing the data item’s key is sent to the node currently responsible for that key. For that purpose,
each forwarding node simply checks its local route cache to determine the node that has the closest known
hash key to the data item’s key and the message is then forwarded to that node. This process continues at
each forwarding node until the packet reaches a node whose own hash key is numerically closer to the data
item’s key than the hash keys of all other nodes in its local route cache. In that case, the node considers
itself responsible for the data item and stores the advertisement (received advertisement message) or replies
(received request message) with a previously stored advertisement. It is quite obvious that there is no way
in which KELOP can guarantee any deterministic routing convergence since it strongly depends on the
content of the routing tables. For example, nodes who’s routing tables are empty will consider themselves
responsible for every data item. In addition to the fact that KELOP does not provide deterministic routing,
the accumulated route to an eventually wrong target node can also significantly deviate from an optimal
direct path.

ISPRP [8] is concerned with initiating a Chord-like overlay ring among the nodes of a MANET without
maintaining a complex structure. The general idea is that each node assigns itself an overlay ID. Each node
only stores the overlay IDs of its one-hop neighbors as well as the overlay IDs of its numerical successor and
predecessor node in the overlay ID space and maintains a DSR style source route to both its successor and
predecessor. Using the successor and predecessor, ISPRP can guarantee the convergence of its key-based
overlay routings. However, a key lookup could take up to \( \frac{n}{2} \) overlay hops (with \( n \) being the number of
nodes in the network) in the worst case.
MADPastry [32] is a DHT explicitly designed for MANETs. MADPastry considers physical locality and integrates the functionality of a conventional DHT and a reactive routing protocol at the network layer to provide an efficient key-based routing primitive in MANETs. MADPastry uses Random Landmarking to map the physical topology to its overlay topology. MADPastry constructs clusters of physically close nodes that share a common overlay ID prefix. Therefore, physically close nodes in MADPastry are also quite likely to be close to each other in the overlay ID space (share a common ID prefix). When employing Pastry on such an overlay topology, in every key lookup, the first few overlay hops are likely to be physically close due to Pastry’s locality awareness mechanism. The rest of the hops are likely to be physically close because the nodes share a common overlay ID prefix.

6.4 Geographical Knowledge Based Location Services

Geographical location services provide the physical location of any given node in the MANET. A location query provides the ID of the requested node and a location reply provides the physical location of the requested node. Location Service and Data Location Service are closely related. Implementing a location service using geographical knowledge has been suggested in GCLP [29]. A comprehensive overview of the protocol is given in Section 5.3. Other geographical quorum-based location services are multi-zone routing [3], GLS [19] and LLS [1]. They deploy a hierarchical structure of the quorum nodes and guarantee the quorum’s intersection.

The position based multi-zone routing [3] 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 $2^{i+1}$. Within a $2^i$ zone, location update is flooded to all of the nodes. The lookup process tracks the location information in a hierarchical manner; while getting closer to a destination, the information about the position of the destination becomes more precise.

GLS [19] 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). Every node has a designated hashed location server within each square it belongs to, based on the node’s id. 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.

The Locality aware Location Service (LLS) [1] uses a hierarchical spiral structure to guarantee the quorums’ intersection. In the basic spiral algorithm, the location of each node is published on a spiral that spans increasingly large squares in the hierarchy. Likewise, searches for each node are performed in increasing spirals on the same virtual hierarchy. The lookup and publishing paths are guaranteed to intersect at the first hierarchy level in which the squares containing the source and the destination intersect. This structure bears resemblance to the hierarchical grid of GLS [19]. However, GLS uses hashing in addition to the geographic hierarchy.

6.5 Other

A field theoretic approach for service discovery protocols is described in [18]. In their approach, scalar fields are defined on the network over which service queries are forwarded towards service instances. A scalar field is analogous to a potential in electrostatics resulting from electrical point charges. The potentials of point charges define a distribution with maxima at the point charges. Analogously, they consider the capacity of service (CoS) as a point charge Q, defining a scalar field on the network with peaks at nodes hosting service instances. The charges that contribute to a potential must be of the same service type. When
When a client searches for a service, it specifies in the request the desired service type. The query is routed to a service instance along the gradient on the appropriate scalar field induced by the CoS of that service type.

Using some sort of caching has been suggested by [10, 15, 27, 31] in order to better locate a data or a service in a MANET. Although incurring some sort of data or service replication, they successfully decrease the amount of traffic induced.

In RINGS [17], nodes publish their data items by flooding the network. However only nodes \((K \cdot n)\)-hops away from the advertising node cache the published data location information where \(K\) is a small constant (e.g. 4) and \(n = 1, 2, 3, \ldots\). The data location information is stored in rings, \(K\) hops apart, around the data owner. Each lookup query has to be flooded to the \((K/2)\)-hops neighborhood of the source node in order to guarantee intersection with at least one of the nodes that cached the data location information. In PDI [20], data location advertisements undergo epidemic dissemination. Thus, by exploiting node mobility PDI can resolve most queries locally without sending messages outside the radio coverage of the inquiring node.

Finally, a comprehensive survey on location services in wireless ad hoc and hybrid networks can be found in [12].

### 7 Discussion and Conclusions

In this paper, we have presented 3DLS, a Density-Driven Data Location Service for ad hoc networks. We first presented the innovative density-driven virtual topography substrate and its utilizable properties. We then showed how such a substrate can be utilized in order to implement a light weight, highly scalable and self adaptive Data Location Service.

The density-driven virtual topography provides a form of random landmarking and centralization mechanism to a completely decentralized environment. We believe that new efficient protocols for various purposes (e.g., routing, broadcasting, etc.) can be designed on top of the density-driven virtual topography substrate. For example, the virtual topography can be used in order to construct and maintain clusters of physically close nodes (much like MADPastry [32] constructs and maintains). This can be done by having each hilltop act as a cluster head and having each node belonging to the same cluster as its hilltop.

Given the cost of broadcast flooding and routing in MANETs, we designed a DLS that totally refrains from both without assuming any geographic knowledge among its nodes. 3DLS is self adaptive and dynamically reacts to changes in the network, namely, its size, its density, nodes’ speed, etc. This is done while keeping the success rate as high as possible in the face of such changes. The adaptivity of 3DLS is made possible by employing an internal feedback mechanism, which records the effort made by previous successful lookups in order to locate their requested data item. These recordings are used to adjust the efforts made by future advertisements accordingly.

Our analysis and simulation results (Section 5) demonstrate that 3DLS is indeed a viable and efficient DLS. By comparing 3DLS to GCLP [29], which assumes geographical knowledge among its nodes, we have shown that 3DLS outperforms a protocol with stronger assumptions regarding the network. We have also compared 3DLS to a hypothetical perfect DHT. In particular, we have shown that the length of advertisement/lookup/reply paths in such a perfect DHT is at least in the order of the average hop distance between two random nodes. The length of each lookup/reply walk in 3DLS is found empirically to be a constant.

The replication level of 3DLS is bounded by the percentage of hilltops among all of the nodes in the network, which is shown empirically to be at most 3\% in well connected MANETs (Section 3.2). This means that in the worst case 3\% of the nodes hold all of the unexpired ads in the network. The worst case in mobile networks (where hilltop switches are frequent) is that each node holds all of the unexpired ads for
an average of 3% of the time. Indeed, each node should be ready at any time to allocate enough memory to store all of the unexpired ads in the network. However, in practice, at least 97% of the time this memory can be used by each node for other purposes, e.g., caching, paging, etc.

Looking ahead, the same principles behind 3DLS can be used to develop other services such as publish/subscribe [9] and tuple spaces [14]. Specifically, in publish/subscribe, publishers post tagged events, whereas subscribers register to receive subscriptions, which specify the type of event the subscriber is interested in. A publish/subscribe service matches between events and subscriptions, and deliver each event to the subscribers that are interested in it. Whenever the events are fairly short, such as in Twitter, one could use the mechanism of 3DLS in the following way: Each event can be propagated with a 3DLS-like advertisement message. On the other hand, a subscription is recorded by the local implementation of the subscriber. From that point on, the subscription is sent as a 3DLS-like lookup message, trying to find all the matchings it can along a lookup walk. Alternatively, one could take the opposite approach, especially when events are large: That is, have each subscription be propagated as a 3DLS-like advertisement message and each event publication as a lookup request. In this case, the handling of a lookup request is to find all interested subscribers of the event, and initiated a message to them. Exploring these options is left for future work. As for tuple spaces, the creation of an object can be done using a 3DLS-like advertisement message, whereas all other operations on the object can be done using 3DLS-like lookup message.
References


