# Energy-Efficient Location Service Protocols for Mobile Ad Hoc Networks

Zijian Wang, Eyuphan Bulut, and Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY, 12180.
{wangz,bulute,szymansk}@cs.rpi.edu

February 27, 2011

**Abstract**

Location-based routing protocols are stateless since they rely on position information in forwarding decisions. However, their efficiency depends on performance of location services which provide the position information of the desired destination node. Although several location service schemes have been proposed, their main goal is merely to find the location of the destination node. Seldom they consider energy efficiency in their designs for the forwarding of location update and query packets. Based on the analysis of the previous works, we propose two novel location service protocols aiming to decrease the distance traveled by the location update and query packets and, thus, to reduce the overall energy cost. Simulation results for both static and mobile networks are presented to demonstrate that the proposed schemes achieve significantly higher energy efficiency and improve the query success rate in comparison to the previously proposed algorithms.

## 1 Introduction

In mobile ad hoc networks, one of the fundamental challenges is to design routing protocols under constantly changing topology. Recently, location based routing has received much attention and is considered to be the most efficient and scalable routing paradigm [1]. However, before a packet can be routed, the source node needs to retrieve the location information of the destination node. Thus, a critical issue for location based routing protocols is to design efficient location services that can track the locations of mobile nodes and at any time reply to queries about the locations of nodes residing anywhere in the network.

### 1.1 Related work

There have been various protocols proposed for location service. The earliest of them were flooding-based approaches. DREAM [2], DLS, and SLS [3] are examples of those in which each node periodically floods the entire network with

its location information. However, the storage and dissemination overhead of such an approach is very high. Reactive flooding-based approaches (e.g., RLS [3]) are better than pro-active ones in terms of overhead. Yet, they might still resort to flooding the entire network when the destination location information is not available in neighbor nodes.

To restrict the location update and query flooding, quorum-based protocols were proposed. One example is the column-row protocol introduced in [4], where each node periodically propagates its location information in the north-south direction, while any location query is propagated in the east-west direction. In this case, the update and query overhead is much lower than it is in flooding-based methods. Yet, the location update cost in terms of hop count is still the full diameter of the network and the query cost could be nearly as high if the query enters the query column far from the intersection of this column with the update row. This method is then extended by sending query and update in non-vertical directions [5] and in multi-directions [6].

Recently, hashing-based protocols, in which location servers are determined via a global hash function, have been proposed. These protocols can further be divided into flat or hierarchical, depending on how the home regions of the location servers are structured. In the flat hashing-based protocols [7, 8], each node's identifier is mapped to a home region consisting of one or more nodes within a fixed location in the network area. All nodes in the home region serve as location servers maintaining location information and replying to location queries. However, there are several drawbacks of such an approach. First, a large overhead is introduced when moving nodes periodically send location updates to their location servers which may be far away. Second, even if the destination node is arbitrarily close to the source node, the source node still needs to send location query to the destination node's location server that could be far away. Third, when all the location servers are within a fixed geographical area, frequent location queries and replies drain energy and cause early death of the nodes within this area. Multi-home region method [9] was proposed to fix some of the above drawbacks.

In the hierarchical hashing-based protocols [10, 11, 12], the network area is recursively divided into a hierarchy of squares. For each node, one or more nodes in each square at each level of the hierarchy are assigned as its location servers. Maintaining a hierarchy offers several benefits. First, moving nodes do not need to send location update to location servers of certain level if they have not moved out of the corresponding square. Thus, the location update cost is significantly reduced. Second, if the source node and the destination node are close to each other and within the same low level square, the location query can be replied quickly. Third, location servers are scattered all over the network, balancing the total network energy usage among nodes.

Since the algorithms that we introduce here take this approach, we summarize the two different phases involved: (i) location query and response, and (ii) location maintenance.

(i) location query and response: When a source node wants to send a packet to a destination node, it sends a location query packet to retrieve the location of the destination node from its location servers. Once any of the location servers for the destination node is found, the location query packet is recursively forwarded to lower level location servers in the lower level squares until the level-1 location server is reached. Finally, the destination node will receive the location query packet and will reply to it with its accurate location. Clearly, the most important step in this procedure is to find the proper location servers.

(ii) location maintenance: Each node maps its ID into a unique location in the unit size square, using a hash function known to all nodes. Using this location at square in which it resides at each level $1 \leq i \leq N$ of the grid hierarchy, the node locates its level-$i$ location server, which simply is the node closest to that unique location. If the node moves from its last reported position further than a predefined distance but remains within its current level-1 square, it sends location update with its exact location information only to its level-1 location server. If the move involves higher level squares, handover procedure is used to inform the relevant servers about the change. The details of processing are further discussed in the subsequent sections.

### 1.2 Contributions

Although energy-related parameters are considered in some routing protocols such as [13], location service protocols seldom consider energy efficiency issue when forwarding location update and location query packets. They focus on the ability to find the location of the destination nodes. In this paper, we address the need for energy efficiency consideration in the design of location service protocols and present novel schemes with low energy consumption. Accordingly, the main contributions of this paper are as follows:

1. Efficient schemes for reducing the distance traveled by location update and query packets are proposed, which lower the energy cost of location service, increase the delivery ratio, and balance the location service load equally among all nodes.

2. The positions of the location servers are optimized to fully minimize the energy consumption and maximize the delivery ratio. It also decreases the number of servers in the network but still balances location service load among nodes.

3. Extensive simulations were performed and results with many varying parameters are presented to show the advantage of proposed schemes over the leading existing protocol in many different environments.

The remainder of the paper is organized as follows. First, we describe the network model, assumptions, and hierarchical coordinate system used by the protocols in Section 2. In Section 3, we present our novel location service schemes. Section 4 provides the simulation results and compares our schemes with the method presented in [12]. We conclude the paper in Section 5.

## 2 Preliminaries

### 2.1 Network Model and Assumptions

We model a mobile ad hoc network as a set of wireless nodes deployed randomly with uniform distribution over a predetermined finite two-dimensional square area. Each node has a unique ID, and is equipped with a communication radio with a communication protocol supporting reliable inter-node communication with adjustable transmission range. We assume that each node knows its own position (e.g., via low power GPS devices or localization techniques [16, 17])

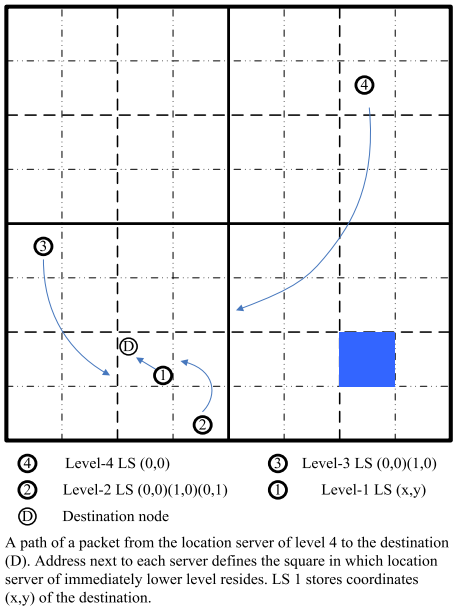



Figure 1: An example for a 4-level hierarchy network.

and also knows the positions of its neighbors. The latter is typically accomplished via periodic hello messages with Time-To-Live (TTL) set to one hop. Thus, this packet will only be received by one-hop neighbor of the sender, instead of flooding the entire network. Additionally, we assume that the nodes move within the square network area according to a mobility model.

## 2.2 Hierarchical Coordinate System

The whole network area is recursively divided into a hierarchy of squares which are known to each node in the network. At the top level, the entire area is called a level-$N$ square, where $N$ is the total number of levels in the hierarchy. Each of level-$i$ ($1<i\leq N$) square is further divided into four level-($i$-1) quadrants, until the entire region is divided into $n = 4^{N-1}$ level-1 squares. Given $L$ as the side length of the whole network area, the side length of a level-$i$ square is $L_i = \frac{L}{2^{N-i}}$. Fig. 1 illustrates an example of a 4-level hierarchy network in which each node resides within exactly one square at each level $i$, such that $1\leq i\leq N$.

Using the lower left point as the origin of the system, we can define the address of level-$i$ square as a sequence of coordinate pairs $(a_x^{N-1}, a_y^{N-1})\ldots(a_x^i, a_y^i)$ (in short $a_{x|y}^i$) computed as:

$$a_{x|y}^i = \frac{s_{x|y}^i - \sum_{k=1}^{N-i-1} L_{N-k} \times a_{x|y}^{N-k}}{L_i} \tag{1}$$

where $(s_x^i, s_y^i)(s_{x|y}^i$ in short) is the lower left coordinate of the level-$i$ square. For example, the address sequence for the marked level-1 square in Fig. 1 is (1,0)(1,0)(0,1).

Inversely, the lower left coordinate of the level-$i$ square can be computed as follows:

$$s^i_{x|y} = \sum_{k=1}^{N-i} L_{N-k} \times a^{N-k}_{x|y} \tag{2}$$

With such a partitioning and the square address scheme applied to the entire network, the specific location of a node can be identified by the square in which this node resides.

Given a node's coordinate ($n_x$, $n_y$), the address sequence ($na^{N-1}_x$, $na^{N-1}_y$)...($na^i_x$, $na^i_y$) (in short $na^i_{x|y}$) of the level-$i$ square to which this node belongs is calculated using the following formula:

$$na^i_{x|y} = \lfloor \frac{n_{x|y} - \sum_{k=1}^{N-i-1} L_{N-k} \times na^{N-k}_{x|y}}{L_i} \rfloor \tag{3}$$

For example, the address sequence of the level-1 square in which the destination node in Fig. 1 resides is (0,0)(1,0)(0,1).

# 3 Energy Efficient Location Service Protocol

In this section, we give the details of how the location update and query operations are performed in the proposed protocol. We also present an analysis which finds the optimal positions of location servers in the current setting.

## 3.1 Location Update

The following key issues need to be addressed in attempt to reduce the distance traveled by the location update packets:

### 3.1.1 Location Server Selection and Update

Each node selects one level-$i$ location server in each level-$i$ square in which it resides using its unique ID and a hash function known to all nodes. Therefore, each node only needs to maintain $N$ location servers. Moreover, the storage overhead is evenly distributed all over the network as nodes with different IDs use different servers. The position of the level-$i$ location server ($ls^i_x$, $ls^i_y$) (referred to as location server point) for each node in level-$i$ square is determined as:

$$(ls^i_x, ls^i_y) = (s^i_x, s^i_y) + hash(ID, L_i) \tag{4}$$

where $ID$ is the unique identifier of the node and ($s^i_x$, $s^i_y$) is the lower left coordinate of the level-$i$ square in which the node resides. $hash(ID, L_i)$ is a global function known to each node that maps a node's $ID$ to a relative position in a level-$i$ square[1]. There may be no node at the exact location server point. In such a case, we choose the node

[1]In the simulations, we used the following simple hash function. For level-$k$ ($1 \leq k \leq N$ for ADJ method and $k = 1$ for OPT method, as explained later) square, we divided it into $M \times M$ grids, where $M \gg n$, then chose the center of grid $(g_x, g_y)$ as the hash point where $g_x = ID\%M$, $g_y = \lfloor ID/M \rfloor$. However, our methods can also use other hash functions.

nearest to the location server point as the corresponding location server using the perimeter based scheme presented in [12].

If the node moves from its last reported position further than a predefined distance but remains within its current level-1 square, it sends location update with its exact location information only to its level-1 location server. Otherwise, if the node moves off the current level-$i$ ($i \geq 1$) square $S_i^{old}$ into new level-$i$ square $S_i^{new}$ within the lowest level-$k$ ($k \geq i+1$) common square $S_k^{com}$ that contains both $S_i^{new}$ and $S_i^{old}$, it updates its location information as follows. First, it sends location update to its level-$k$ location servers. Second, it sends location update to all of its level-$j$ ($1 \leq j \leq i$) location servers in $S_i^{new}$. Third, it sends location remove packets[2] to all of its outdated level-$j$ ($1 \leq j \leq i$) location servers in $S_i^{old}$.

Obviously, if a node oscillates between two nearby points at two sides of a high level square boundary, sending of location updates immediately after each slight location change will be costly. Therefore, to reduce such an overhead, we employ lazy update technique similar to ones presented in [14, 15], and let each node send location update only if it moves out of level-$i$ square for at least a certain threshold distance $d(L_i)$.

### 3.1.2 Location Information Update and Storage

In the proposed method, each location server maintains a list of nodes whose location information it stores. Each element of the list stores the following information: node $ID$ (32 bits), location server level ($log_2 N$ bits), location information (will be introduced in the next paragraph), and expiration time (32 bits).

It should be noted that the exact location information of destination nodes is only stored at level-1 location servers. At all other levels, the location servers only store the address sequence of the square in which the level-($i$-1) location server (and also the destination node) resides, as shown in Fig. 1. There are three advantages of storing location information in this way. First, the memory usage is reduced because the address sequence of a square takes only 2($N$-$i$+1) bits for level-$i$ location server while the exact location information takes 64 bits. For each location server, on average, there are only $N$ entries[3] in the list. That is, for the example in Fig. 1, where $N = 4$, the memory usage is only 340 bits[4] per node. Second, the size of the location update packet is also reduced which decreases the energy cost for location update. Third, the location information at level-$i$ location server needs to be updated only when the destination node moves out of the corresponding level-($i$-1) square, which significantly reduces the frequency of location updates and thus the energy consumption.

[2]Here, both the location update and remove packets are sent following the route computed by greedy Hamiltonian path algorithm. We elaborate on this in Section 3.1.4

[3]Since there are $N$ location servers for each node in the network, in total we have $N\times$(node count) entries in the tables of all nodes. This makes an average of $N$ entries per node (or location server) in the network.

[4]On average, each node becomes a level-$i$ server for only one node. Therefore, it keeps 130 bits for the node for whom it serves as level-1 server and 68, 70 and 72 bits for the nodes for whom it is level-2,-3 and -4 server, respectively.

### 3.1.3 Location Information Handover

Each location server periodically (with the same frequency of hello messages for all the nodes) checks each entry in its list and calculates the distance between its current position and the location server point (computed by Eq. 4) for each destination node. If this distance exceeds certain predefined handover threshold, the current location server will choose the neighbor node closest to the corresponding location server point as the new location server[5]. With the sufficiently large move of a location server between the checking times, it is also possible that it can lose its 'location server duty' for more than one nodes at a time. Therefore, the location server may need to inform multiple new location servers (each for a different node) about such loss. Even in such cases, only one location handover packet is broadcast to accomplish that. The packet carries a list of location servers to be informed (indexed by the server's node ID) and the corresponding location information to be stored at each server. Each node receiving location handover packet checks whether it is on the list. If this is the case, it stores the corresponding location information. Compared to the broadcasting of location handover packet for each new location server individually, this solution decreases the chance of packet collision (an observation confirmed by simulation) and consequently reduces the energy cost.

### 3.1.4 Sending Location Update Packet

In previous work, all the location update packets are sent to location servers individually. In our protocol, we achieve location updates in a more energy efficient way. If one node needs to send a location update to more than one location server, it first calculates the distances that would be traveled by the update messages in two cases: (i) when they are sent to each desired location server individually (referred as d-indiv) (ii) when they are sent in one packet that traverses all the desired location servers (referred as d-all). If d-indiv incurs smaller distance than d-all does, the location update messages are sent to each desired location server individually. Otherwise, all the update messages are integrated into one packet that is forwarded according to a forwarding table indicating the sequence of location servers to be visited. Traversing multiple points in a plane is an instance of the Hamiltonian path problem. We use a simple greedy solution in which the next visited node is always the nearest one to the currently visited node. Any intermediate node greedily forwards location update packet to the neighbor nearest to the position of the next location server in the forwarding table. Once the location update packet reaches a location server at certain level, the corresponding location information will be stored at this server and the next entry in the forwarding table will pop up. If certain intermediate node can not find a neighbor node closer to the location server in the current forwarding table entry than itself, this entry will be dropped and the next table entry will pop up. All the outdated table entries are deleted and therefore the corresponding server will not be updated[6].

In order to reduce packet size, only the table entry for level-1 location server stores the exact location information of the destination node (64 bits). All the other table entries store only location server level ($\lceil \log_2 i \rceil$ bits) and the

[5]Note that this handover procedure involves only the old and new location servers of a node and it is different than the update procedure defined in Section 3.1.1. Both procedures indeed run in parallel. Therefore, it is also possible that even if a node does not move, its level-$i$ location server may change due to the movement of current level-$i$ location server.

[6]This lowers the location query success rate, but it happens so infrequently that its impact on the performance of the protocols is negligible.

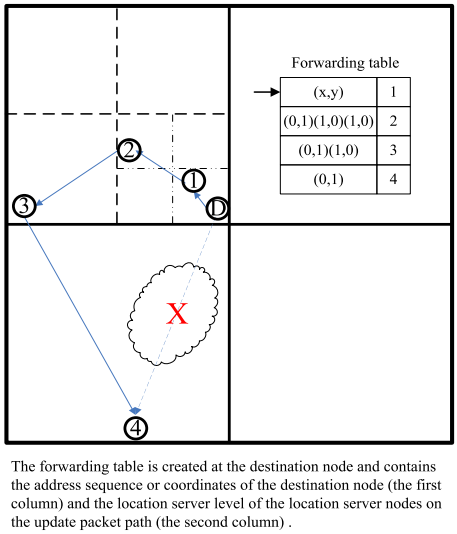



Figure 2: An example of location update and forwarding table.

address sequence for the level-(i-1) square in which the destination node resides ($2(N\text{-}i{+}1)$ bits). The computational complexity of this coding procedure is $O(N^2)$. Note that, when all $n$ nodes are evenly distributed over the entire network which is divided into $n = 4^{N-1}$ squares, we obtain $N \approx \log(n)/2$. Thus the computational complexity of the coding procedure indeed becomes $O(\log(n)^2)$. Any intermediate node receiving the location update packet can decode the information to get the location of the server in the current forwarding table entry.

The first entry in the forwarding table in Fig. 2 is an example. Any intermediate node can get the address sequence of the level-1 square in which the level-1 location server resides. This address is computed from the destination node's location ($x$,$y$) by applying Eq. 3. Then, the lower left coordinate of the square can be computed by applying Eq. 2. Finally, the position of the corresponding level-1 location server can be calculated using Eq. 4. The computational complexity of this decoding procedure is $O(N^2)$ ($O(\log(n)^2)$).

An advantage of sending location update in one packet instead of many is that the distance traveled by the location update packet is shorter, reducing the energy cost.

## 3.2 Location Query

Here, we focus on the most important step, location query processing, that is used to find the proper location servers to obtain location information.

### 3.2.1 Observations and Basic Idea

We made the following observations about the previous methods described in [12, 13, 14]. In the method introduced in [12] (referred to as HIGH-GRADE method and abbreviated as HG in our paper), the source node calculates all candidate level-$i$ location server points assuming the destination node resides in the same level-$i$ square as itself. Then,

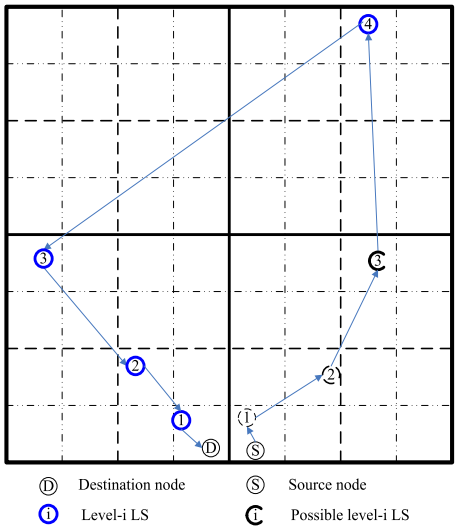



Figure 3: Location query scheme from [12].

the location query packet traverses the candidate location server points in increasing order of location server levels until the lowest level square in which both the source and the destination nodes reside is found. Clearly, such a common square always exists (in the worst case this is the level-$N$ square). The main drawbacks of this method are as follows. First, the real location server could be quite nearby, but the location query packet has to travel a long distance to find it. One example of such a situation is shown in Fig. 3. Second, the location query packets are always forwarded from lower to higher levels of candidate location server points, even if visiting the latter and dropping the former will decrease the distance traveled by packet. In fact, if the high level candidate location server is not a real one, then neither is the low level one. If the location query packet can check the high level candidate location server points first, then there is no need to check the low level candidate location server points at all. Thus, both the distance traveled by the location query packet and the corresponding energy cost could be reduced.

Schemes proposed in [14, 15] tried to address the aforementioned first drawback by forwarding location query packet in a spiral with increasing radius until it meets one of the location servers. Consequently, the nearby location servers can be found quickly. Yet, the location query packet may still travel a long distance if the location servers are far away from the source node.

We observed that:

1. for the source node, it is worth searching the adjacent squares outside its own high level square, but only if the expected gain (finding right location server quickly, thus decreasing the average distance traveled by packet) is bigger than the cost for visiting extra location server points;

2. if jumping over lower level candidate location server points and visiting higher level candidate location server point first will decrease the average distance traveled by packet, then the source node should send the location query packet to visit the higher ones first.

### 3.2.2 Location Query Procedure

Based on our observation, we propose a new location query method (referred to as ADJ), as shown in Algorithm 1. We first analyze the gain and cost of using this new method and then introduce the location query procedure in detail.

**Gain and cost analysis**

If the source node finds (with probability of $4^{-N+k}$) the right (e.g. with information about the destination) level-$k$ location server when searching adjacent squares (these servers are called extra location servers), then we gain by avoiding sending first a query packet to a sequence of servers at levels growing from 1 to $N$ and then descending from $N$ to $k$. Hence, the gain measured in distance is:

$$g_k = \left( d(source, LSP_1) + \sum_{i=1}^{N-1} d(LSP_i, LSP_{i+1}) + \sum_{i=k}^{N-1} d(LSP_i, LSP_{i+1}) \right) 4^{-N+k} \tag{5}$$

where $d(p_i, p_j)$ denotes the distance from $p_i$ to $p_j$.

**Algorithm 1** Location Query Procedure

```
1: Determine which method to use (ADJ or HG)
2: Find optimal visiting list sequence
3: if ADJ is used then
4:    Find adjacent squares to be searched
5:    Compute gain and cost for each nearby square found
6:    if gain > cost then
7:       Put corresponding location server point into visiting list
8:    end if
9: end if
```

If the source node visits one level-$k$ location server point ($LSP_{k_n}$) in adjacent square but does not find the destination location information there (this happens with probability of $1-4^{-N+k}$), then the location query packet has to go back to visit possible location servers within source node's square using HG method (referred to as base location servers). In this case, we get no gain but pay the extra cost. Assume $LSP_1$ is the first location server point that the source node will visit using HG method. Then the cost will be:

$$\begin{aligned} c_k =& (4^{-N+k}) d(source, LSP_{k_n}) + \\ & (1 - 4^{-N+k}) \left[ d(source, LSP_{k_n}) + \right. \\ & \left. d(LSP_{k_n}, LSP_1) - d(source, LSP_1) \right] \\ =& d(source, LSP_{k_n}) + \\ & (1 - 4^{-N+k}) [d(LSP_{k_n}, LSP_1) - d(source, LSP_1)] \end{aligned} \tag{6}$$

**Selecting the method to use**

When the source node wants to find the location of the destination node, it first draws a circle with itself as center with the estimated maximum gain as a radius. If this circle intersects with other level-$h$ (predefined parameter, should be high, we set it to $N$-1) squares (not the one containing the source node), the source node will choose to use ADJ method. Otherwise, the source node will choose to use HG method. The maximum gain is estimated as follows. It is clear that $max(distance(source, LSP_1)) = \sqrt{2}L_1$, where $L_1$ is the side length of level-1 square. According to the hash function we used, $d(LSP_{i+1,i+2}) = 2d(LSP_i, LSP_{i+1})$. Given the ID of the destination node and a level-2 square, it is easy to compute the exact maximum distance from the four possible $LSP_1$s to $LSP_2$ (referred to as $L(1,2)_{max}$). Thus, Eq. 5 becomes:

$$g_k = \frac{\left(\sqrt{2}L_1 + L(1,2)_{max}\left(\sum_{i=1}^{N-1} 2^{i-1} + \sum_{i=k}^{N-1} 2^{i-1}\right)\right)}{4^{N-k}} \tag{7}$$

It is easy to prove (see Appendix) that $g_k$ is a strictly increasing function of $k$, thus we have the maximum $g_k$ when $k = N - 1$:

$$g_{k_{max}} = \left(\sqrt{2}L_1 + L(1,2)_{max}((3(2^{N-2}) - 1)\right)/4$$

**Finding optimal visiting list sequence**

**Algorithm 2** Optimal Visiting List Sequence

```
1: opt_path = 2^{N-1} - 1
2: opt_dist = Length(opt_path)
3: for i = 0; i < 2^{N-1} - 1; i++ do
4:    dist = Length(i)
5:    if dist < opt_dist then
6:       opt_dist = dist
7:       opt_path = i
8:    end if
9: end for
```

In original HG method, the source node visits servers from $LSP_1$ to $LSP_N$ in sequence. However, in our protocol, if the node selects to run HG method, we send the query over the optimal visiting list sequence (path) giving the minimum cost among all possible paths (there are $2^{N-1}$ of them) from source to $LSP_N$. For example, if the source node visits $LSP_1$ and $LSP_2$ in sequence, then with probability of $4^{-N+1}$, it will find the destination location information in $LSP_1$ and stop going further; with probability of $1 - 4^{-N+1}$, it will continue to search $LSP_2$. Thus the average cost of getting from source to $LSP_2$ is $d(source, LSP_1) + (1 - 4^{-N+1})d(LSP_1, LSP_2)$. If the source node drops $LSP_1$ and visits $LSP_2$ directly, then with probability of $4^{-N+2}$, it will find the destination location information in $LSP_2$ and will go to the level-1 location server that contains detailed destination location information. Since only one out of four servers at level 1 serviced by $LSP_2$ is $LSP_1$, the probability that the search will go back from $LSP_2$ to $LSP_1$ is just $1/4 \times 4^{-N+2}$, or $4^{-N+1}$. With probability of $1 - 4^{-N+2}$, the search will continue to server $LSP_3$. Thus the average

cost of getting from the source to $LSP_2$ (when jumped over $LSP_1$) is $d(source, LSP2)+(4^{-N+1})d(LSP_2, LSP_1)$. In general, optimal visiting list sequence finding process is shown in Algorithms 2 and 3.

**Finding adjacent squares to be searched**

In order to narrow down the possible adjacent squares to be searched, we use the following process. After finding optimal path, source node knows the exact cost from itself to $LSP_N$ (Let $L(s, N)$ denote this cost). It then draws a new circle with itself as center and with a radius of $r_{est}=\left[L(s,N) + L(1,2)_{max}\left(\sum_{i=k}^{N-1} 2^{i-1}\right)\right] 4^{-N+k}$. If this circle intersects with level-$k$ ($1 \leq k \leq h$) squares contained within another level-$h$ square (not the one that contains the source node), the source node will put the corresponding level-$k$ squares into list. For each of the level-$k$ squares in the list, the source node will calculate $LSP_k$ assuming the destination node is within this square. If $LSP_k$ is not within the new circle, the corresponding square will be removed from the list.

**Algorithm 3** Length(int path)

```
1: dist = 0
2: last_node = 0
3: for i = 1; i ≤ N; i++ do
4:   if ((i^th digit of path from right is 1) || (i = N)) then
5:     if (last_node=0) then
6:       dist = dist + d(LSP_last_node, LSP_i)
7:     else
8:       dist = dist + d(LSP_last_node, LSP_i) × (1 − 4^(−N+last_node))
9:     end if
10:    last_node = i
11:  else
12:    dist = dist + d(LSP_i, LSP_i+1) × 4^(−N+i)
13:  end if
14: end for
15: return dist
```

**Gain and cost comparison**

For each square in the final adjacent square list, the source node will compute the exact gain and cost using Eq. 5 and Eq. 6. If the gain exceeds the cost, the corresponding location server point in adjacent square will be put into the visiting list.

Fig. 4 shows an example in which the source node resides in the level-1 square which is beside the boundary of level-3 square. After refining the four base location server points, the level-1 location server point is removed. The source node finds some extra location server points in adjacent squares, but only the level-3 location server point in adjacent square (0,0) will be put into the visiting list. Then, all candidate location server points are sorted in the order shown in Fig. 4, as defined by the path originating in the source node and traversing all nodes in the list.

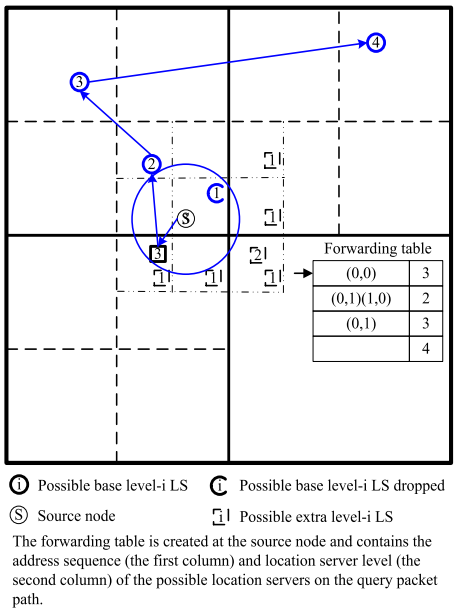



Figure 4: Location query procedure and forwarding table.

To reduce the packet size, the forwarding table uses the same information coding technique which was used by the location update procedure. All table entries store the location server level $i$ ($\lceil log_2 i \rceil$ bits) and the address sequence of the level-$i$ square in which the candidate location server resides ($2(N-i)$ bits). Any intermediate node receiving the location query packet can decode the information and learn the location of the candidate location server in the current forwarding table entry. During the location query procedure, if any location server with destination information is found, the location query packet will stop following the forwarding table and instead will use the information found in this location server.

Consider the forwarding table shown in Fig. 4. From the first table entry, an intermediate node calculates the lower left coordinate of the square by applying Eq. 2 to address sequence (0,0). Then, the position of the corresponding candidate location server can be calculated using Eq. 4 that is of complexity $O(N)$ ($O(\log(n))$).

### 3.3 Location Server Position Optimization

In the previous section, we tried to reduce the location update and query costs by adjusting the paths traveled by these packets. In this section, we try to reduce the location update and query costs by adjusting the position of location servers. We first take a two-level grid as an example to analyze the average cost of location query procedure and then derive the optimal position for location servers.

As shown in Fig. 5, the two-level grid is divided into four level-1 grids, marked as grids $A$, $B$, $C$ and $D$. $H_{1a}$, $H_{1b}$, $H_{1c}$ and $H_{1d}$ are four level-1 location server points for these grids. Note that the relative position to the lower left corner of level-1 grid for each of these points is the same for the same destination node, thus the distance between $H_{1a}$ and $H_{1b}$ is $L_1$, which is the side length of the level-1 grid. $H_2$ is the location server point for the level-2 grid.

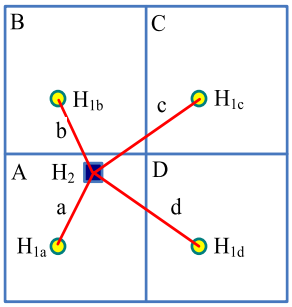



Figure 5: A two-level grid example.

We denote the distance between $H_{1a}$ and $H_2$ as $a$, the distance between $H_{1b}$ and $H_2$ as $b$, the distance between $H_{1c}$ and $H_2$ as $c$, and finally the distance between $H_{1d}$ and $H_2$ as $d$. The source node and the destination node could fall within each level-1 grid with equal possibility of 1/4. Thus, in total, there are 16 cases, each of them occurring with equal probability of 1/16, as shown in Table 1. Since the same procedure will apply to the lower levels, we ignore the distance traveled by the location query packet between the source node (or the destination node) to the level-1 location server in the same level-1 grid and just take the distance between level-1 location server and level-2 location server into consideration. For example, if the source node and the destination node are within the same grid $A$, then the location query packet will first go to $H_{1a}$ and then will find the destination node's information and go to the destination node directly. Thus, the location query cost will be 0. If the source node is within grid $A$ and the destination node is within grid $B$, then the location query packet will visit $H_{1a}$, $H_2$, $H_{1b}$ and the destination node in sequence. Thus the location query cost will be $a$+$b$. By considering all possibilities, we conclude that the average location query cost is $\frac{3(a+b+c+d)}{8}$, as shown in Table 1.

| Source/Destination | A | B | C | D |
|---|---|---|---|---|
| A | 0 | a+b | a+c | a+d |
| B | b+a | 0 | b+c | b+d |
| C | c+a | c+b | 0 | c+d |
| D | d+a | d+b | d+c | 0 |

Table 1: Location query cost analysis.

We can minimize average location query cost by adjusting the position of $H_2$. We take the case that the source node is in grid $A$ and the destination node is in grid $D$ as an example to derive the optimum position of $H_2$. Let's assume that we decide to find the best position for server $H_2$ at certain distance $x$ from one side of square $H_{1a}H_{1b}H_{1c}H_{1d}$. As shown in Fig. 6, given the arbitrary line $H'H_2H$", parallel to side $H_{1a}H_{1d}$, $H_{1am}$ is the mirror image of $H_{1a}$ over this line. Hence, $H_{1d}H_2 + H_2H_{1am} > H_{1d}H_{1am}$ by triangle principle, then $H_{2-opt}$ projected onto side $H_{1a}H_{1d}$ (point $H_{2p}$) lays in the middle of this side, because triangle $H_{1d}H_{2-opt}H$" is equal to triangle $H_{1am}H_{2-opt}H'$. Placing a mirror on line $H_{2p}H_{2-opt}$ shows that the optimal placement of server $H_2$ is at the center of square of potential positions of

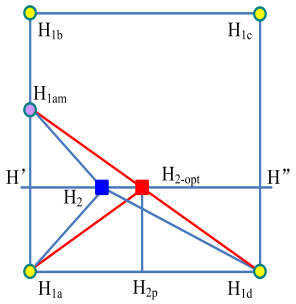



Figure 6: Optimal placement of server $H_{l+1}$.

servers at level 1.

Hence, given the position $(x_1, y_1)$ of the level-1 location server in the most lower-left level-1 grid within the whole grid, the optimum position of level-$i$ $(x_i, y_i)$ location server could be computed as:

$$x_i = x_1 + \sum_{k=1}^{i-1} \frac{L_k}{2} \text{ and } y_i = y_1 + \sum_{k=1}^{i-1} \frac{L_k}{2}$$

**Algorithm 4** Average Location Query Cost

1: $dis = 0$
2: **for** $x_1 = 0.5/M$; $x_1 < L_1$; $x_1 = x_1 + 1.0/M$ **do**
3:  **for** $y_1 = 0.5/M$; $y_1 < L_1$; $y_1 = y_1 + 1.0/M$ **do**
4:  $dis = dis + \sqrt{x_1^2 + y_1^2}$
5:  $dis = dis + \sqrt{(L_1 - x_1)^2 + y_1^2}$
6:  $dis = dis + \sqrt{x_1^2 + (L_1 - y_1)^2}$
7:  $dis = dis + \sqrt{(L_1 - x_1)^2 + (L_1 - y_1)^2}$
8:  **end for**
9: **end for**
10: $dis = \frac{3dis}{8(M^2 L_1^2)}$

We can also estimate the benefit from the optimization of location server positions. When the location servers are placed at the optimal positions, then $a$=$b$=$c$=$d$=$\sqrt{2}L_1/2$ in Table 1, thus the average cost of location query will be $1.0607L_1$. Moreover, when the locations of servers are determined by Eq. 4, we can compute the average location query cost by using numerical integration[7] as shown in Algorithm 4, that yields the average distance of $1.1478L_1$ (increasing $M$ improves accuracy). Thus, we can expect to achieve an improvement of about 8.2%.

According to the location update rule introduced above, the average location update cost will be $\frac{a+b+c+d}{4}$, which is 2/3 of the average location query cost. Thus, we can get the same improvement for location update when the location

[7]Assuming that an $M \times M$ grid is located on the whole area and the location servers can be in the middle of each of these grid cells with the same probability, we compute the average location query cost in all possible cases. Note that, if $H_{1a}$ is at point $(x_1, y_1)$, $H_2$ is located at $(2x_1, 2y_1)$. Then the sum of (a+b+c+d) in Fig. 5 for a single case becomes the sum computed in lines 4-7 of Algorithm 4.

servers are placed at the optimized positions.

# 4 Simulations

## 4.1 Simulation Model and Settings

We used NS-2.33 simulator to evaluate our proposed schemes and compared them with the HIGH-GRADE method presented in [12] (referred to as HG). Our method that adjusts the paths traveled by location update and query packets is referred to as ADJ, while our method with the location servers optimally placed is referred to as OPT. It should be noted that OPT shares with ADJ all other protocol improvements introduced above (however, they may yield different results due to the different positions of servers versus themselves and the source node). The whole network is deployed over a 1000 m by 1000 m area partitioned into 4-level squares. IEEE 802.11 is used as the MAC and physical layer protocol. We used two-ray-ground propagation model. Each node's transmission range varies from 0 to $R_{max}$. The power consumed by each transmission is 1.6 W for omni-directional transmission of the maximum 250 m range and lower for shorter transmit ranges. The power drained for reception is constant and equal to 1.2 W. The detailed energy consumption model for nodes with ranges less than $R_{max}$ could be found in NS2 documentations [18].

## 4.2 Performance Metrics

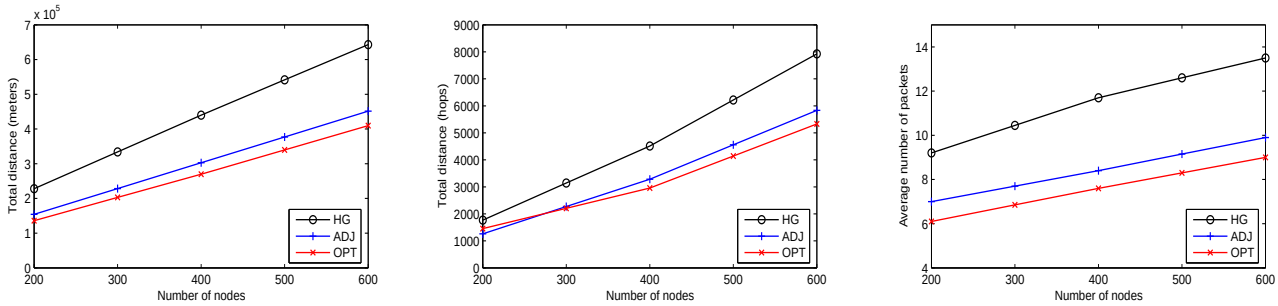
To evaluate the performance of the proposed schemes, we used the following metrics:

1. The average total distance[8] traveled by all location update packets for all nodes, referred to as *update distance*.
2. The average number of packets forwarded by each node, referred to as *packet count*.
3. The average distance traveled by location query packets, referred to as *query distance*.
4. The average delay of location query packets, referred to as *query delay*.
5. The average energy usage[9] per node in the network.
6. The average location query success rate.

The distance traveled by a location update or query packet is accumulated during the packet forwarding procedure. For example, if a packet is forwarded from node $A$ to node $B$, its hop distance will increase by one and its traveled distance will increase by the distance between node $A$ and node $B$.

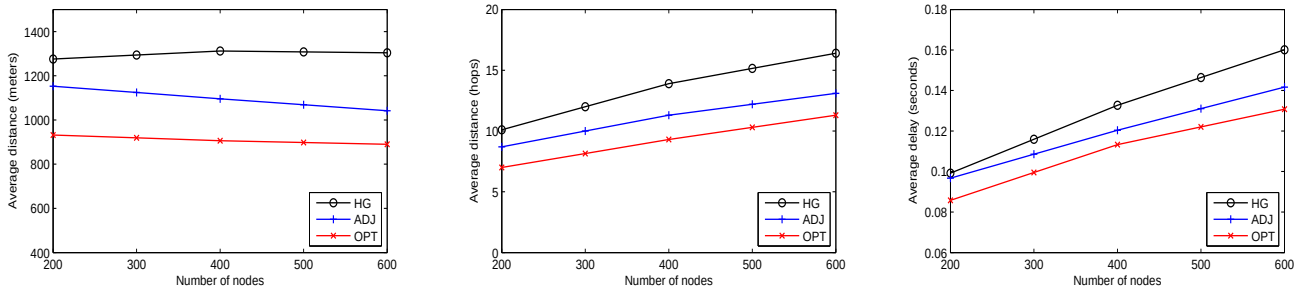
[8] We sum the total distance traveled by all location update packets for all nodes in a single topology, then take the average of this sum for different topologies. Distance is measured both in meters and hops. We provide different graphs for each.

[9] Since all the methods compared use periodic hello messages, the cost of those messages is not included in total energy usage.




(a) Metric 1 (in meters) (b) Metric 1 (in hops) (c) Metric 2

Figure 7: Performance comparison of three algorithms according to metrics 1 and 2 in static network scenario.




(a) Metric 3 (in meters) (b) Metric 3 (in hops) (c) Metric 4

Figure 8: Performance comparison of three algorithms according to metrics 3 and 4 in static network scenario.

## 4.3 Simulation Results

This section presents the results of our evaluations of the proposed algorithms according to the aforementioned metrics. We used both the static and mobile network scenarios. Although many other location service studies do not look at the performance of their algorithms for static networks, we demonstrate them here for the sake of completeness and for showing the superiority of our algorithms even in the static case.

### 4.3.1 Static Network Scenario

In this scenario, we keep the neighbor count for each node constant and vary the number of nodes in the network from 200 to 600 by changing $R_{max}$ from 177 m to 102 m[10]. This enables us to evaluate the proposed schemes with different numbers of nodes in the network while preserving the number of neighbors of each node. We randomly generated five static topologies for each configuration. For each of them we ran 20 groups of simulations. Each simulation ran for $S + 45$ seconds, where $S$ is the number of nodes. For the first $S$ seconds, node $s$ sent location update at $s^{th}$ second[11]. Then, starting from $(S + 20)^{th}$ second, each of five randomly selected source nodes generated five location queries to randomly selected destination node, with 5 seconds interval between its queries.

[10]These settings are similar to the settings of [12], where the compared algorithm, HG, is presented.

[11]Here, we applied this scheme to avoid packet collision in the initial location update procedure. When nodes start to move, as in the scenarios considered in the next section, they will send location updates only when necessary.

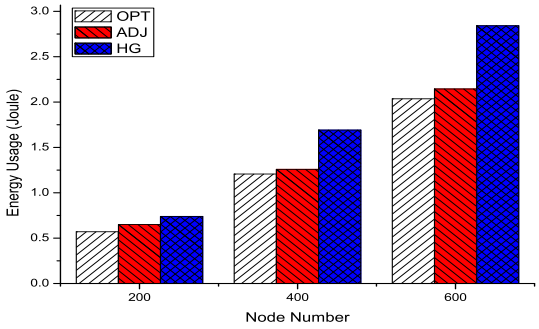



Figure 9: Average energy usage per node in static network scenario.

Figure 7 presents the average results (of all topologies) for update distance and packet count as a function of number of nodes. Clearly, the location update costs for our two methods are much lower than for HG. This is mainly because the location update messages in our method could be sent in one packet. The location update cost for OPT is even lower than for ADJ because the distance traveled by the location update packets is further reduced by adjusting the positions of location servers. On average, we get improvement of 9.6%, which is very close to the analysis result 8.2% in Section 3.3. Since most of the packets forwarded are location update packets and the average number of forwarded packets by each node is directly proportional to the hop distance, the slopes of the plots (and the relations between the slopes of plots) shown in Figure 7c look similar to the ones in Figure 7a and 7b, which also verifies the simulation results in these figures.

Figure 8 gives the average results for query distance and delay as a function of number of nodes. The cost of the ADJ is lower than HG, with an average improvement of 17%. The improvement comes from two scenarios: 1) the source node finds the destination location information in adjacent squares worthy visiting; 2) the visiting list is improved by Algorithm 2. Consequently, ADJ method decreases the location query delay in worst case too. This is important for some time sensitive applications, which requires that there is a limit for the location query delay in worst case. Moreover, the cost of OPT method is much lower than ADJ and HG because the optimized positions of location servers provide additional cost savings. Furthermore, since the delay of location query packets is directly proportional to the hop distance of location query packets, the slope of simulation results shown in Fig. 8c is very similar to the slopes of results in Figures 8a and 8b, which also verifies the simulation results in these figures.

We also computed the energy usage for three methods. The results are illustrated in Fig. 9. As expected, the energy usage for ADJ is lower than HG, and it is in the range of 75% to 87% of HG's energy use. The energy usage for OPT is even lower than ADJ, in the range of 88% to 95% of the ADJ's energy use. Consequently, ADJ and OPT can provide 13% to 25% and 24% to 30% saving against HG, respectively.

Table 2 shows the average location server counts for three methods in static network scenario. We can see that ADJ has almost the same location server number as HG, but OPT has much fewer location servers. In Fig. 10, we illustrate the average remaining energy (and its variance (mean square error)) of location server nodes and non-server

nodes with respect to time for three methods in static scenario[12]. From the figure, we observe that the energy usage for location server nodes are a little higher than for non-server nodes in all three methods. This is because the location servers spend their energy on two activities. First is receiving location update, receiving location query and sending location reply for itself. Second is forwarding location update, location query and location reply for other nodes. In contrast, non-server nodes only forward location update, location query and location reply. OPT uses fewer location servers than HG, suggesting that the work and energy usage per each location server will be higher in OPT than in HG. Yet, the energy usage (and variance) for location server nodes and non-server nodes are nearly the same in ADJ and OPT[13]. Both of them, however, are lower than in HG, which means that 1) our two methods use less energy and have better energy balancing than HG; 2) the energy usage is evenly distributed in OPT even though its location server count is only 45% to 75% of location servers in ADJ. From these simulation results, we conclude that most of the energy is spent by forwarding packets for other nodes, and therefore evenly distributed among all nodes.

| node count | 200 | 400 | 600 |
|---|---|---|---|
| HG | 128.8 | 274.4 | 546 |
| ADJ | 125.2 | 274.4 | 546 |
| OPT | 99 | 175.2 | 251.3 |

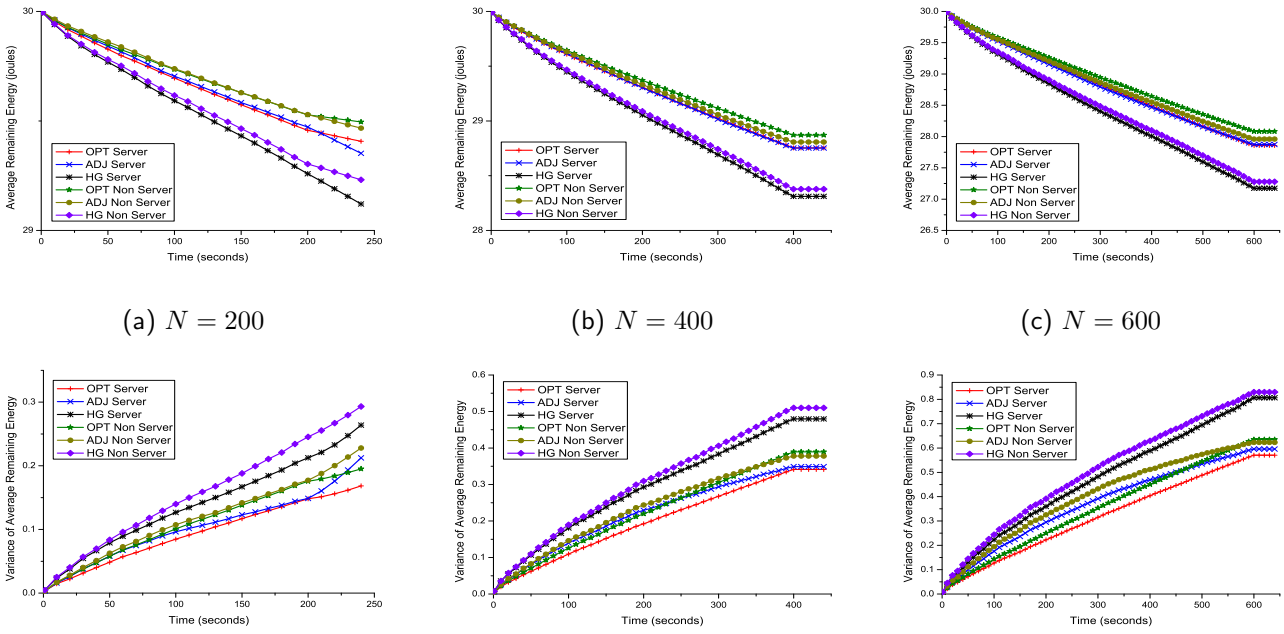
Table 2: Average location server count in static network scenario.

Here, we also would like to discuss why the energy usage (and variance) for location server nodes and non-server nodes in ADJ and OPT are almost the same, even though OPT uses more location server. Consider Fig. 11 where we show the increasing sharing of the higher level servers in OPT versus ADJ. Location server at the first level in OPT are selected equally randomly as in ADJ method. Location servers at the second level in OPT must be within unit square (yellow) while in ADJ they are randomly placed within the 4 unit square with yellow sides. Thus, OPT level 2 servers are 4 times more likely to be on the same node as ADJ servers. Location servers at the third level in OPT must be within unit square (red) while in ADJ they are randomly placed within the 16 unit square with red sides. Thus, OPT level 3 servers are 16 times more likely to be on the same node as ADJ servers. Location servers at the fourth level in OPT must be within unit square (black) while in ADJ they are randomly placed within the whole are with black sides of 64 units. Thus, OPT level 4 servers are 64 times more likely to be on the same node as ADJ servers.

Note that this is easy to prove from the fact that level $k$+1 servers are the centers of squares with level $k$ servers, so they are level $k$ square side from in each direction from each level $k$ server. For example, the whole lowest left square (that contains all possible location of level-1 server in this square) shifts half a unit up and right to create yellow square. Then, yellow square moves up and right a unit to create red square. Finally red square moves up and right 2 units to become black square. That proves that if two nodes share level-1 server in OPT, they share all higher level servers, which is not true in ADJ.

[12]Because location update cost is much larger than location query cost, we only consider location update energy usage here.

[13]This is because in OPT average distance between two subsequent levels of location servers is shorter than in ADJ, so packets sent to higher level servers travel shorter distance in OPT than in ADJ. We explain this in more detail in Appendix.




(a) $N = 200$ (b) $N = 400$ (c) $N = 600$

(d) $N = 200$ (e) $N = 400$ (f) $N = 600$

Figure 10: Average remaining energy and variance in static scenario.

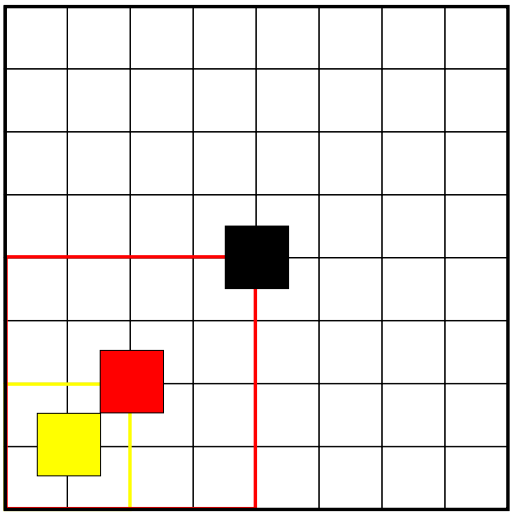


Figure 11: Possible location server regions in ADJ and OPT. While the shaded (yellow, red and black) unit squares show the regions for OPT, the square regions with colorful sides show the regions for ADJ.

Consequently, if a node is in colored squares, it has higher chance of becoming higher level server in OPT then in ADJ, but this loss is moderated by the gain of sending its packets to servers in shorter distance than in HG, as they are close (because packets are sent directly to higher level servers, especially if they are close). Conversely, if it is not in color squares it can never be a high level server (it can be at most level 1 server) but since it is far from servers, and since the packets are needed to be sent to higher level servers directly often, while gaining on one part, it loses on the other. Because of all these reasons, in OPT there is still a good balancing of energy usage and the energy usage (and variance) for location server nodes and non-server nodes in ADJ and OPT are almost the same.

As a final statistics in static scenario, in Table 3 we show the average location query success rate for three methods. We observe that OPT method yields higher success rate than ADJ does, while HG method has the lowest success rate. This is mainly because of the higher number of packets forwarded by each node in HG method than in ADJ and OPT methods (as it is illustrated in Figure 7c), which increases the chance of packet collision resulting in location update or query packet loss.

| Node count | 200 | 400 | 600 |
|---|---|---|---|
| HG | 93.5% | 92.9% | 78.0% |
| ADJ | 98.2% | 95.6% | 83.3% |
| OPT | 99.2% | 96.8% | 86.1% |

Table 3: Average location query success rate in static network scenario.

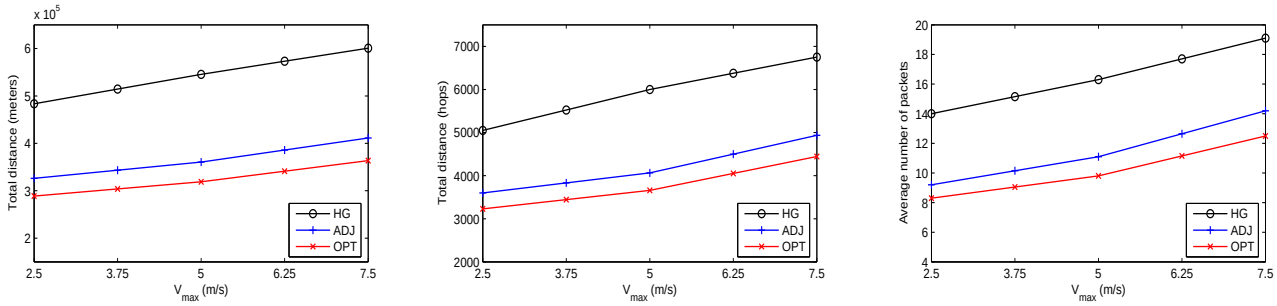
#### 4.3.2 Mobile Network Scenario

In the mobile network scenario, nodes move according to the random way-point[14] mobility model with no pause time. The moving speed for each node is chosen between zero and a maximum moving speed $V_{max}$. We keep the number of nodes in the network at 400 but vary the maximum nodal speed $V_{max}$ from 2.5 m/s to 7.5 m/s. All the other configurations are the same as for the static network scenario, except that all nodes start to move after $(S + 20)^{th}$ second. The $d(L_i)$ used in lazy update procedure is set to $\frac{L_i}{20}i$ and the handover threshold is set to 90 m[15].

Figure 12 plots the average simulation results for update distance and packet count as a function of maximum speed $V_{max}$. The plots show that the location update cost (both hop count and distance traveled) grows for all methods with the increase of $V_{max}$. This is because when the maximum speed of nodes increases, nodes need to send location update packets more frequently (as they move out of a certain level of the grid), which will increase the location update cost. But it is much lower for both of our methods than for HG for two reasons. First, our methods send location update in one packet. Second, we use the lazy update procedure which reduces the update cost when node oscillates near the square boundaries.

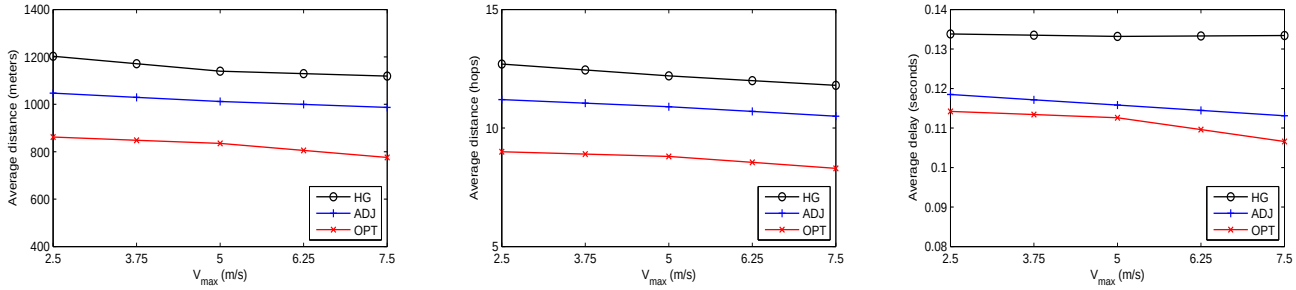
[14]We also performed simulations using several other mobility models such as random walk and random direction. Since the results were similar to the ones with random way-point model, we did not include them in the paper for the sake of brevity.

[15]We chose these values after an extensive run of simulations with different values.




(a) Metric 1 (in meters) (b) Metric 1 (in hops) (c) Metric 2

Figure 12: Performance comparison of three algorithms according to metrics 1 and 2 in mobile network scenario.




(a) Metric 3 (in meters) (b) Metric 3 (in hops) (c) Metric 4

Figure 13: Performance comparison of three algorithms according to metrics 3 and 4 in mobile network scenario.

Figure 13 shows the average simulation results for query distance and delay as a function of $V_{max}$. As the plots illustrate, the cost of ADJ is lower than HG and the cost of OPT is lower than ADJ for the same reasons that we discussed in case of the network scenario illustrated in Fig. 8. It is interesting to note that the location query cost decreases with the increased maximum speed for all methods but for different reasons. The location query packet in HG is always forwarded to level-1 location server first and then forwarded to immediately higher level location servers. Thus, the decrease of location query cost comes only from the increase of node's mobility. For ADJ and OPT, besides the reason for HG method, the chance of meeting higher level location servers for each node increases with the increase of $V_{max}$. According to the location query scheme of ADJ and OPT introduced above, the location query packets will be forwarded to higher level location servers directly, which reduces the distance traveled by these packets.

The energy usage for mobile scenario is shown in Fig. 14. As expected, the energy usage grows with the increase of speed $V_{max}$. This is because the main energy cost results from location update, which increases with the increase of maximum node speed. However, ADJ method uses only 70% of the energy used by the HG method, while OPT method uses even less of it. For example, OPT uses 95% of the energy used by the ADJ when $V_{max}$=7.5 m/s.

Table 4 shows the average location server counts used in each method in mobile network scenario (we still count a node as location server after it sends location handover packet and transfers server duty to other nodes). Again, ADJ has almost the same number of location servers as HG does, but OPT uses much fewer of them. Fig. 15 shows the average remaining energy (and its variance (mean square error)) of location servers and non-servers with time for

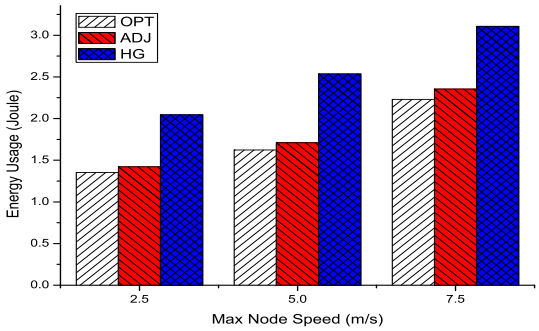



Figure 14: Average energy usage per node in mobile network scenario.

three methods in mobile scenario. We observe that the difference between the variance of server nodes and non-server nodes for each of the three methods decreases with the increase of node speed, which indicates that the mobility helps balancing the energy usage among all nodes [19]. It is also clear that the energy usage (and variance) for all nodes in ADJ and OPT are almost the same, which means that the energy usage is evenly distributed in OPT even though it uses 20% to 40% fewer location servers than ADJ does. The reason for this is the same as in static scenario.

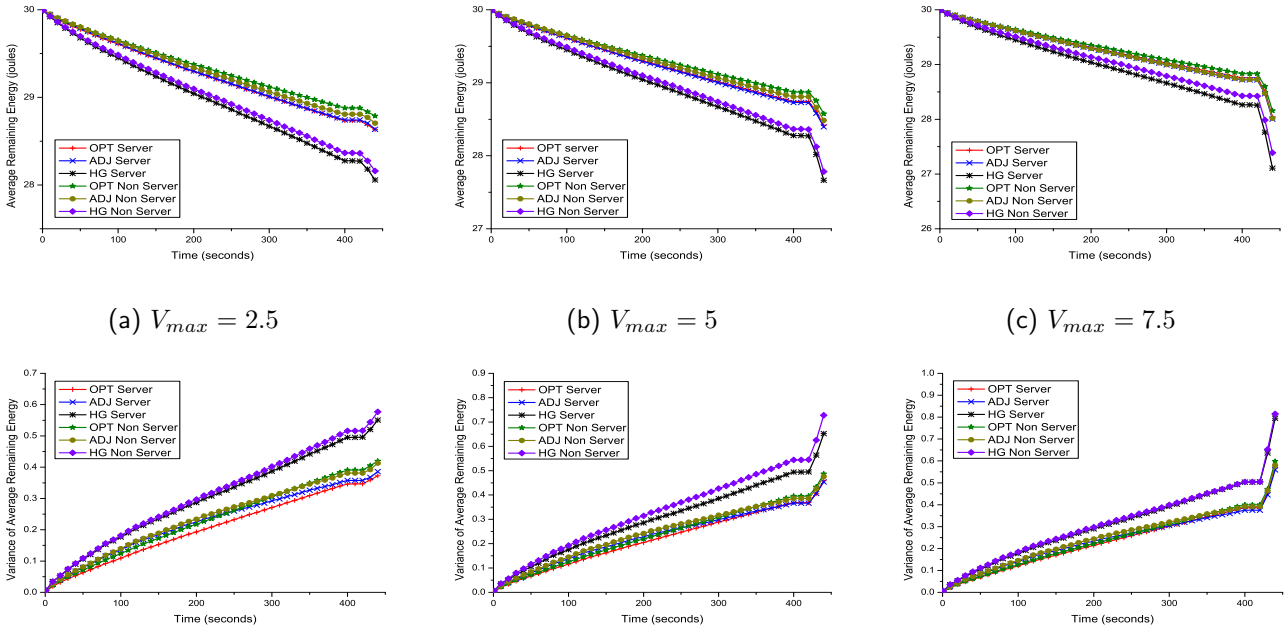
| $V_{max}$ (m/s) | 2.5 | 5 | 7.5 |
|---|---|---|---|
| HG | 296.6 | 326 | 348.4 |
| ADJ | 291.8 | 332 | 359 |
| OPT | 188.2 | 242.2 | 290.4 |

Table 4: Average location server count in mobile network scenario.

Table 5 shows the average location query success rate for three methods in mobile network scenario. They drop with the increase of $V_{max}$ but still OPT has the highest success ratio, while ADJ is better than HG. From Fig. 12c, we see that in HG method each node forwards more packets than ADJ and OPT methods. This increases the chance of packet collision and results in location update or query packet loss, thus a decrease in success rate.

| $V_{max}$ (m/s) | 2.5 | 5 | 7.5 |
|---|---|---|---|
| HG | 62.9% | 48.7% | 41.2% |
| ADJ | 67.5% | 53.5% | 49.5% |
| OPT | 69.7% | 60.8% | 51.0% |

Table 5: Average location query success rate in mobile network scenario.




(a) $V_{max} = 2.5$ (b) $V_{max} = 5$ (c) $V_{max} = 7.5$

(d) $V_{max} = 2.5$ (e) $V_{max} = 5$ (f) $V_{max} = 7.5$

Figure 15: Average remaining energy and variance in mobile scenario.

# 5 Conclusion

In this paper, we introduced two novel location service protocols with the goal of reducing the overall energy cost by decreasing the distance traveled by the location update and query packets. To achieve this goal, one method, ADJ, adjusts the path for location update and query packets, while the other one, OPT, places the location servers at their optimal positions. Extensive simulations were performed to demonstrate that the new schemes achieve significantly higher energy efficiency and improve query success rate when compared to the existing methods.

In future work, we will use these location services in designing routing protocols and applications for mobile wireless networks. We also plan to analyze the effect of utilizing such energy efficient location services in the design of routing protocols such as [20, 21] for delay tolerant networks where the intermittently occurring contacts between nodes and low node density makes the routing challenging. Moreover, we will also look at the problem of finding optimum $N$ (number of hierarchical levels in the network) that provides the best energy efficiency for the given number of nodes in the network and its area of coverage. Furthermore, the influence of real-world environment will also be considered in the future [22].

# Appendix

### The calculation of $p_k$

The probability that one possible location server point of level $k$ ($1 \leq k \leq N$) residing in a level-$k$ square contains information about the destination is equivalent to the probability that the destination resides in the level $k$ square. Hence, this probability is $p_k = 4^{-N+k}$.

### The proof of why $g_k$ is a strictly increasing function of k

$$\begin{aligned} g_k &= \sqrt(2)L_1 + L(1,2)_{max} \times \left( \sum_{i=1}^{N-1} 2^{i-1} + \sum_{i=k}^{N-1} 2^{i-1} \right) \times 4^{-N+k} \\ &> L(1,2)_{max} \times \left( \sum_{i=1}^{N-1} 2^{i-1} \right) \times 4^{-N+k} \\ &> L(1,2)_{max} \times 2^{N-1} \times 4^{-N+k} \\ &= L(1,2)_{max} \times 2^{2k-N-1} \end{aligned}$$

and

$$\begin{aligned} g_{k+1} &= \sqrt(2)L_1 + L(1,2)_{max} \times \left( \sum_{i=1}^{N-1} 2^{i-1} + \sum_{i=k+1}^{N-1} 2^{i-1} \right) \times 4^{-N+k+1} \\ &= 4g_k - L(1,2)_{max} \times 2^{k-1} \times 4^{-N+k+1} \end{aligned}$$

Thus,

$$\begin{aligned} g_{k+1} - g_k &= 3g_k - L(1,2)_{max} \times 2^{k-1} \times 4^{-N+k+1} \\ &> 3L(1,2)_{max} \times 2^{2k-N-1} - L(1,2)_{max} \times 2^{-2N+3k+1} \\ &> L(1,2)_{max} \times 2^{2k-N} - L(1,2)_{max} \times 2^{-2N+3k+1} \\ &= L(1,2)_{max} \times 2^{2k-N} \times \left(1 - 2^{k+1-N}\right). \end{aligned}$$

Because $k+1 \leq N$, thus we have $g_k \leq g_{k+1}$.

# Acknowledgment

Research was sponsored by US Army Research Laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

# References


[1] R. Friedman and G. Kliot, *Location Services in Wireless Ad Hoc and Hybrid Networks: A Survey*, Technical Report CS-2006-10, Technion - Israel Institute of Technology, April, 2006.

[2] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward, *A distance routing effect algorithm for mobility (DREAM)*, in Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom), 1998, pp. 76-84.

[3] T. Camp, J. Boleng, and L. Wilcox, *Location information services in mobile ad hoc networks*, in Proceedings of IEEE International Conference on Communications, 2002, pp. 3318-3324.

[4] D. Liu, I. Stojmenovic, and X. H. Jia, *A scalable quorum based location service in ad hoc and sensor networks*, in Proceedings of IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS), 2006, pp. 489-492.

[5] F. Yu, Y. Choi, S. Park, E. Lee, M. S. Jin, and S.H. Kim, *Sink Location Service for Geographic Routing in Wireless Sensor Networks*, In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), 2008; pp. 2111-2116.

[6] H. Jeon, K. Park, D.-J. Hwang, and H. Choo,*Sink-oriented Dynamic Location Service Protocol for Mobile Sinks with an Energy Efficient Grid-Based Approach*, Sensors, Vol 9, No. 3, pp. 1433-1453, 2009.

[7] S. C. Woo, and S. Singh, *Scalable routing protocol for ad hoc networks*, ACM Wireless Networks, Vol. 7, No.5, pp. 513-529, 2001.

[8] S. M. Das, H. Pucha, and Y. C. Hu, *Performance comparison of scalable location services for geographic ad hoc routing*, in Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2005, pp. 1228-1239.

[9] B.-C. Seet, Y. Pan, W.-J. Hsu. and C.-T. Lau, *Multi-Home Region Location Service for Wireless Ad Hoc Networks: An Adaptive Demand-driven Approach*, in Proceedings of the Second Annual Conference on Wireless On-demand Network Systems and Services (WONS), pp. 258-263, 2005.

[10] Y. Yan, B. X. Zhang, H. T. Mouftah, and J. Ma, *Hierarchical location service for large scale wireless sensor networks with mobile sinks*, in Proceedings of IEEE Global Telecommunications Conference (GLOBECOM), 2007, pp. 1222-1226.

[11] S. Ahmed, G. C. Karmakar, and J. Kamruzzaman, *Hierarchical adaptive location service protocol for mobile ad hoc network*, in Proceedings of IEEE Wireless Communications and Networking Conference (WCNC), 2009, pp. 1-6.

[12] Y. Z. Yu, G.-H. Lu, and Z.-L. Zhang, *Enhancing location service scalability with HIGH-GRADE*, in Proceedings of IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), 2004, pp. 164-173.

[13] L. Cao, T. Dahlberg, and Y. Wang, *Performance evaluation of energy efficient ad hoc routing protocols*, in Proceedings of IEEE International Performance, Computing, and Communications Conference (IPCCC), 2007, pp. 306-313.

[14] I. Abraham, D. Dolev, and D. Malkhi, *LLS: a locality aware location service for mobile ad hoc networks*, in Proceedings of Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), 2004, pp. 75-84

[15] R. Flury, and R. Wattenhofer, *MLS: an efficient location service for mobile ad hoc networks*, in Proceedings of 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2006, pp. 226-237.

[16] A. Savvides, C.-C Han, and M. B. Srivastava, *Dynamic fine-grained localization in ad-hoc networks of sensors*, in Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), pp. 166179, 2001.

[17] D. Niculescu and B. R. Badrinath, *Ad Hoc Positioning System (APS) Using AOA*, in Proceedings of INFOCOM, 2003.

[18] http://www.isi.edu/nsnam/ns/

[19] M. Grossglauser and D. N. C. Tse, *Mobility increases the capacity of ad-hoc wireless networks*, in Proceedings of IEEE Infocom, 2001.

[20] E. Bulut, Z. Wang and B. K. Szymanski, *Cost Effective Multi-Period Spraying for Routing in Delay Tolerant Networks*, in IEEE/ACM Transactions on Networking, vol. 18, 2010.

[21] E.Bulut, S. Geyik and B. K. Szymanski, *Efficient Routing in Delay Tolerant Networks with Correlated Node Mobility*, in Proceedings of 7th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Nov, 2010.

[22] S. Ahmed, G. C. Karmakar, and J. Kamruzzaman, *Evaluating Performance of Location Service Protocols of Ad-Hoc Wireless Network in Real-World Environment Model*, in Proceedings of Wireless Communications, Networking and Mobile Computing (WICOM), pp. 1577-1580, 2007.