# An Energy Efficient Dynamic Location Server Hierarchy for Mobile Ad Hoc Networks

Zijian Wang, Eyuphan Bulut, Boleslaw K. Szymanski, Graham Bent, Abbe Mowshowitz, Paul Stone

*Abstract*— **By using position information in forwarding decisions, location based routing protocols for mobile networks are stateless and efficient. However, they are heavily dependent on location services which provide the position information of the desired destination node. Although several location service schemes have been proposed, their main goal is just to find the location of the destination node. Seldom they include energy efficiency metrics when evaluating their performance in forwarding location update and query packets. Based on the analysis of the previous related works, we propose a novel location service for mobile networks that aims at decreasing the distance traveled by the location update and query packets and, thus, at reducing the overall energy cost. The solution uses a hierarchy of mobile servers embedded into the network. As a result, the same scheme can be used for mapping logical networks with desired properties, such as hypercubes, onto to mobile ad hoc networks efficiently. A connection of this solution to the efficient query execution in the Gaian database is also discussed. Simulation results for both static and mobile wireless networks are presented to demonstrate that the new scheme achieves energy efficiency while maintaining all the other performance metrics comparable to the previously published algorithms.**

*Index Terms*—**Ad-hoc Networks, Gaian Database, Location Service, Mobile Networks.**

## I. INTRODUCTION

ONE of the fundamental challenges of mobile ad hoc networks 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. However, before a packet can be routed, the source node needs to retrieve the location 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 reply to location queries throughout the entire network at any time.

Numerous protocols for location service have been proposed. The earliest of them were flooding-based approaches. DREAM

Zijian Wang, Eyuphan Bulut and Boleslaw K. Szymanski are with the Rensselaer Polytechnic Institute, Troy, NY 12180 USA (corresponding author: Boleslaw K. Szymanski, phone: 518-276-2714; fax: 518-276-4033; e-mail: szymansk@cs.rpi.edu).

Graham Bent and Paul Stone are with Emerging Technology Services, IBM UK, Hursley Park, Winchester, Hants, UK.

Abe Mowshowitz is with the City University of New York, New York, NY 10016 USA.

[1], DLS, and SLS [2] are examples of those in which each node periodically floods its location to all network nodes. However, the storage and dissemination overhead of such an approach is very high. Reactive flooding-based approaches (e.g., RLS [2]) are better than proactive flooding-based methods in terms of overhead, but they still require flooding the whole network to find the destination, if it 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 [3], where each node periodically propagates its location information in the north-south direction, while any location query is propagated in the east-west direction. The update and query overhead is much lower than for flooding-based methods, but 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 column does not intersect with the update column shortly.

Recently, hashing-based protocols have been proposed, in which location servers are determined via a global hash function. These protocols can further be divided into flat or hierarchical, depending on the structure of the area used. In the flat hashing-based protocols [4-5], 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 approach. First, a large overhead is introduced when moving nodes periodically send location update 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.

In the hierarchical hashing-based protocols [6-8], 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 chosen 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 near to each other and within the same low level square, the location query can be replied quickly. Third, location servers, which can be mobile, are scattered all over the network, which balances the network energy usage. However, the main goal of the hierarchical hashing-based protocols is just to find the location of the destination nodes. They seldom take energy efficiency issue into consideration when forwarding location update and location query packets. We propose a novel location service scheme which attempts to decrease the distance traveled by the location update and query packets and, thus, to reduce the overall energy cost.

The remainder of this paper is organized as follows. We first describe the model and its assumptions in Section II. In Section III, we present our novel location service scheme. Section IV gives the simulation results comparing our scheme with the method presented in [8]. In section V, a connection of this solution to the efficient query execution in the Gaian database is discussed. We conclude the paper in Section VI.

## II.    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 providing reliable inter-node communication with adjustable transmission range. We assume that each node knows its own position (e.g., via GPS devices) and also knows the positions of its neighbors (this is accomplished typically via broadcasting hello messages periodically). Additionally, we assume that the nodes move around the square network area.

## III.    ENERGY EFFICIENT LOCATION SERVICE

### A.   Network Partition and 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 determined by the total number of levels in the hierarchy. Each of level-$i$ ( $1 < i \le N$ ) squares is further divided into four level-$(i-1)$ quadrants, until the entire region is divided into $4^{(N-1)}$ level-1 squares. Given $L$ as the side length of the whole network area, the side length of a level-$i$ square is $L_i = L/2^{N-i}$. Fig. 1 illustrates an example of a 4-level hierarchy network. In such a hierarchy, each node resides within exactly one square at each level $i$, such that $1 \le i \le N$ .

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 $(a_x^{N-1}, a_y^{N-1})...(a_x^i, a_y^i)$ ( $a_{x|y}^i$ in short) computed as:

$$a_{x|y}^i = (s_{x|y}^i - \sum_{k=1}^{N-i-1} L_{N-k} \bullet a_{x|y}^{N-k}) / L_i \qquad (1)$$

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:

$$s_{x|y}^i = \sum_{k=1}^{N-i} L_{N-k} \bullet a_{x|y}^{N-k} \qquad (2)$$

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 $(na_x^{N-1}, na_y^{N-1})...(na_x^i, na_y^i)$ ( $na_{x|y}^i$ in short) of the level-$i$ square to which this node belongs is calculated as:

$$na_{x|y}^i = floor((n_{x|y} - \sum_{k=1}^{N-i-1} L_{N-k} \bullet na_{x|y}^{N-k}) / L_i) \qquad (3)$$

where $floor(x)$ returns the largest integer not greater than x. 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).
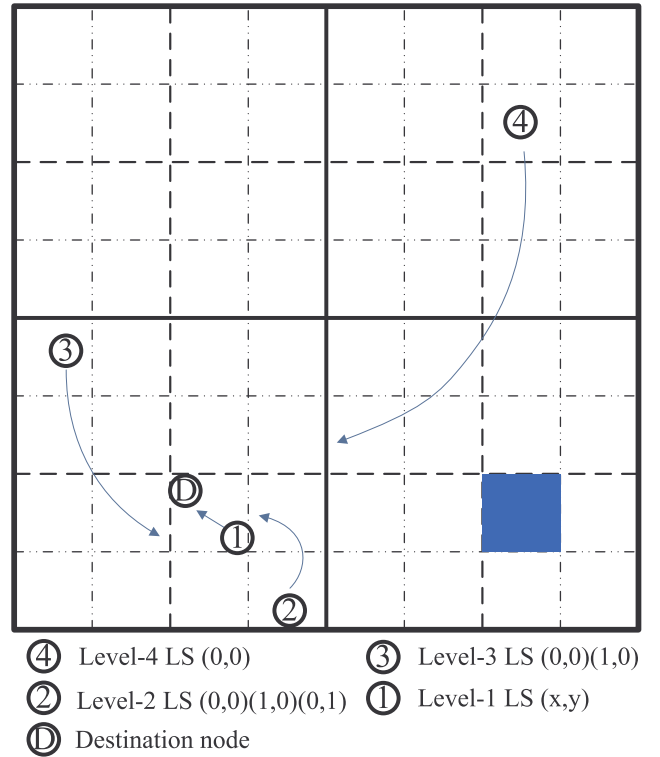


| ④ | Level-4 LS (0,0) | ③ | Level-3 LS (0,0)(1,0) |
| ② | Level-2 LS (0,0)(1,0)(0,1) | ① | Level-1 LS (x,y) |
| Ⓓ | Destination node | | |

**Fig. 1.** An example for a 4-level hierarchy network

### B.   Location Update

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

*Location Service Selection*

Each node selects one level-i location server in each level-i square in which it resides. Therefore, each node only needs to maintain a constant number of location servers. Also the storage overhead is balanced smoothly all over the network. The position of the level-i location server $(ls_x^i, ls_y^i)$ (referred to as location server point) for each node in level-i square is determined as:

$$(ls_x^i, ls_y^i) = (s_x^i, s_y^i) + hash(ID, L_i) \qquad (4)$$

where ID is the unique identifier of the node and $(s_x^i, s_y^i)$ is the lower left coordinate of the level-i square in which the node resides. Hash is a global function known to each node that maps a node's ID to a relative position in a level-i square.

There may be no node at the exact location server point. We choose the node nearest to the location server point as the corresponding location server.

*Location Information Update and Storage*

In our 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 ($log_2N$ bits), location information (introduced in the following), and expiration time (32 bits).

Please note that the destination node's exact location information 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 in the list and the memory usage is only 340 bits. 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, thereby saving a lot of energy.

*Location Information Handover*

Each location server periodically 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. There may be more than one new location server that needs to be informed about this change. Still, 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 so, it will store the corresponding location information. Compared to broadcasting 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.

*Sending Location Update Packet*

In previous methods, all the location update packets are sent to location servers individually. In our method, if one node needs to send a location update to more than one location server, it first calculates the distances traveled by the update messages both for sending them to each desired location server individually (referred as d-indiv) and sending them in one packet which traverse all the desired location servers (referred as d-one). The method with smaller cost is used. One packet update is forwarded according to a forward table which indicates the sequence of location servers to be visited. Traversing multiple points in a plane is a kind of 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 forward 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 forward table pops up. If certain intermediate node cannot find a neighbor node closer to the location server in current forward table entry than itself, the current table entry will be dropped and the next table entry will pop up. All the outdated table entries are deleted.
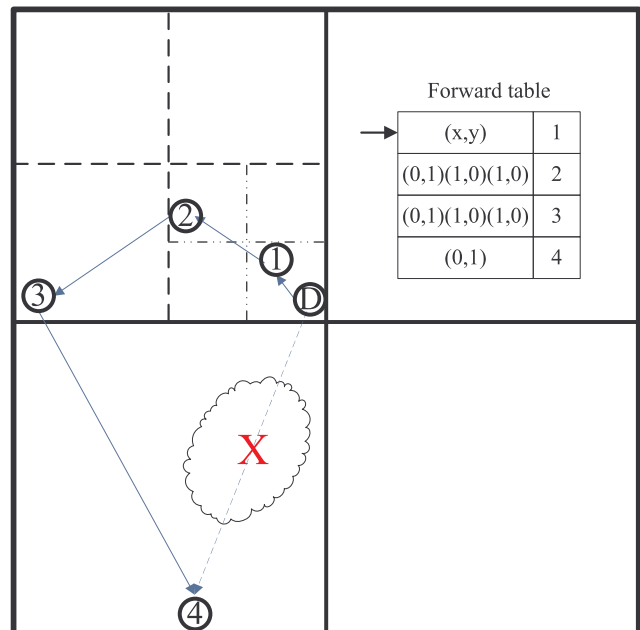


**Fig. 2.** An example of location update and forward table

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 (*ceiling(log₂i)* bits, where *ceiling(x)* returns the smallest integer not less than *x*) and address sequence for the

level-(i-1) square in which the destination node resides (*2(N-i+1)* bits). The computational complexity of this coding procedure is $O(n^2)$. Any intermediate node receiving the location update packet can decode the information to get the location of the location server in the current forward table entry.

The first table entry in the forward 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 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 $O(n^2)$.

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

*Which and When Location Servers Are Updated*
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 from current level-i ( $i \geq 1$ ) square $S_o$ into new level-i square $S_n$ within the lowest level-k ( $k \geq i+1$ ) common square $S_c$ that contains both $S_n$ and $S_o$, it updates its location information as follows. First, it sends location update to all of its level-k location servers. Second, it sends location update to all of its level-j ( $1 \leq j \leq i$ ) location servers in $S_n$. Third, it sends location remove packets to all of its outdated level-j ( $1 \leq j \leq i$ ) location servers in $S_o$.

Requiring that each node sends location update immediately is costly when the node oscillates between two nearby points at two sides of a high level square boundary. Such a node sends location updates to many location servers repeatedly. To reduce such an overhead, we employ lazy update technique similar to one presented in [9-10], in which each node sends location update only if it moves out of level-i square for at least a distance $d(L_i)$.

### C. Location Query

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 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. Thus, the most important step is to find the proper location servers.

*Observations and Basic Idea.*
We made the following observation about the previous methods described in [8-10]. In the method introduced in [8] (referred to as *HIGH-GRADE method* and abbreviated *HGM*),

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, 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 followings. 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. Although the possibility that the instance shown in Fig. 3 happens is low (about 2% of this specific case), the useless distance traveled by the location query packet is very long. Second, the location query packets are always forwarded from lower to higher levels of candidate location server points, even if the later are closer to the source node than the former. 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.
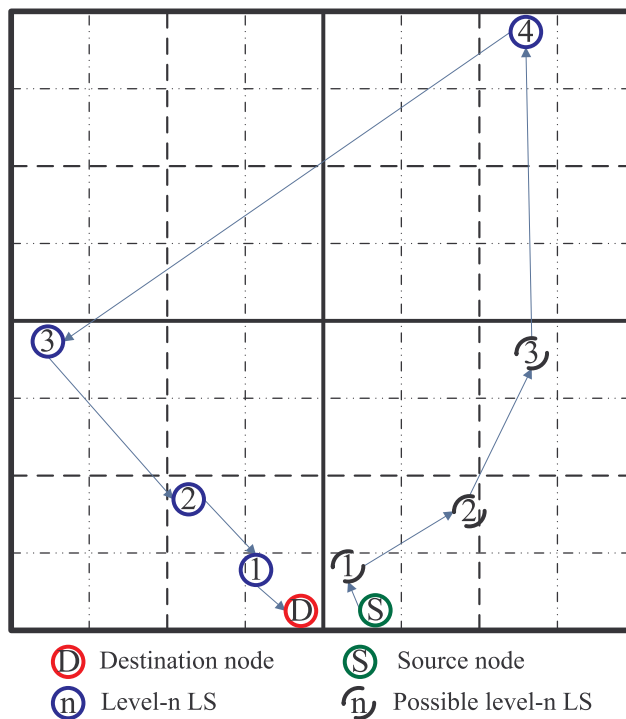


**Fig. 3.** Location query scheme from [8]

Schemes proposed in [9-10] tried to address the first drawback mentioned above 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, still the problem remains that the location query packet may travel a long distance if the location servers are far away from the source node.

We observed that: (1) if the source node resides near the boundary of a high level square, it is worth searching the adjacent squares on the opposite side of the boundary for location servers, but only if the level of these adjacent squares is low (otherwise the extra search forces the location query packet to travel a long distance); (2) if higher level candidate location server point is closer than low level candidate location server point from current node, then this node should send location query packet to visit the former one first.

*Location Query Procedure*

When the source node resides within a level-s (predefined parameter, should be low, we set it to 1) square that is beside the boundary of any level-h (predefined parameter, should be high level, we set it to *N-1*) square, the search proceeds as follows. The source node calculates all candidate location server points (from level-N to level-1) that fall into the adjacent level-s squares (we refer to them as candidate adjacent squares) located on the opposite side of the boundary of a level-h square. If any level-k candidate location server point is found in each adjacent level-s square, there is no need to find level-i (i<k) candidate location server points in the same level-s square. Hence, only the highest candidate location server in each adjacent square searched needs to be found (we refer to it as *extra location server*).
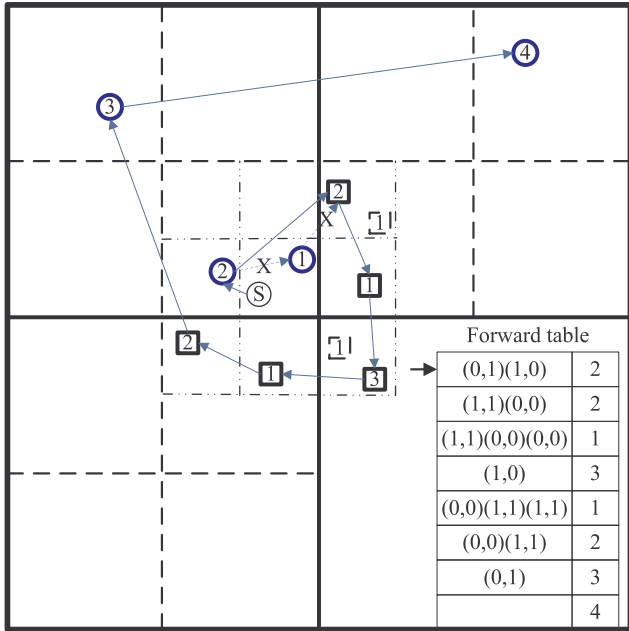


**Fig. 4.** Location query procedure and forward table

Otherwise, the source node will only find all candidate location server points (we refer to them as *base location servers*) using the HIGH-GRADE method. Both of these two kinds of location points (if the extra location servers exist) are sorted into a list in the sequence that can be traversed by one path starting from the source node, using the same greedy Hamiltonian path method as sending location update packets. If any high level base location server is in front of low level base location server, the lower one is deleted from the list. This is because if the location query packet checks the high level base location server points first, then there is no need to check the low level base location server points, as mentioned in our observation. But for extra location servers, we need to check all of them in each adjacent level-s square, so we keep all of them.

An example in which the source node resides in the level-1 square which is beside the boundary of level-3 square is shown in Fig. 4. The source node calculates all candidate location server points in the adjacent squares (there are at most five of them, as shown in Fig. 4). There are two candidate location server points in adjacent level-1 square (1,0)(0,1)(0,1) (one is level-1, the other is level-3). Only the level-3 one will be kept. Then, all candidate location servers are sorted in the order shown in Fig. 4, as defined by the path traversing from the source node. Since the level-2 base location server is in front of level-1 base location server on this sorted list, the level-1 base location server will be removed from the list. The computational complexity of this procedure is $O(n^3)$.

To reduce the packet size, the forward table uses the same information coding technique which was used by the location update procedure. All table entries store the following information: location server level i (*ceiling(log_2 i)* bits), 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 get the location of the candidate location server in the current forward table entry. During the location query procedure, if any real location server is found, the location query packet will stop following the forward table but will be forwarded using the new information from the real location server.

Take the forward table in Fig. 4 as an example. From the first table entry, an intermediate node calculates the lower left coordinate of the square from address sequence (0,1)(1,0) by applying Eq. (2). Then, the position of the corresponding candidate location server can be calculated using Eq. (4). The computational complexity of this procedure is $O(n)$.

## IV. SIMULATIONS

We used NS-2.33 simulator to evaluate our proposed scheme and compared it with the HIGH-GRADE method presented in [8]. 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 following metrics are evaluated for the location service protocols: (1) the total distance (measured in meters and hops) traveled by all location update packets for all nodes; (2) the

average distance traveled by location query packets; (3) the average distance traveled by location query packet for specific destination node (for this kind of location query, we select the source node that resides within a level-1 square that is beside the boundary of level-3 square; moreover, there is at least one location server for the destination node residing in the adjacent level-1 square which is also beside the boundary of level-3 square on the opposite side); (4) the average energy usage; and (5) the location query success ratio.

*Static Network Scenario*

In this scenario, we keep the number of neighbor nodes constant and vary the number of nodes in the network from 200 to 600 by changing $R_{max}$ from 177m to 102 m. We randomly generated 5 static topologies for each configuration and ran the simulation for 420 seconds. In each topology simulation, first all nodes send location update packets, then, 20 randomly selected source nodes generate 20 location queries to randomly selected destination nodes. We also recorded all the location information stored at each node. Thus, we were able to find the desired source-destination pair for specific location query for metric (3).

**Table 1.** Static network results for metric (1)

| Node Number | 200 | 400 | 600 |
|---|---|---|---|
| Our method: distance | 154129 | 302726 | 452977 |
| (hops) | (1259) | (3285) | (5845) |
| Compared method: distance | 227285 | 435528 | 649543 |
| (hops) | (1736) | (4495) | (8037) |

**Table 2.** Static network results for metric (2)

| Node Number | 200 | 400 | 600 |
|---|---|---|---|
| Our method: distance | 1289 | 1316 (13.9) | 1317 |
| (hops) | (9.9) | | (16.6) |
| Compared method: distance | 1299 | 1303 (13.7) | 1366 |
| (hops) | (10.0) | | (17.2) |

**Table 3.** Static network results for metric (3)

| Node Number | 200 | 400 | 600 |
|---|---|---|---|
| Our method: distance | 879 | 967 | 1098 |
| (hops) | (7.2) | (10.3) | (14.4) |
| Compared method: distance | 1444 | 1425 | 1497 |
| (hops) | (11.2) | (14.5) | (18.7) |

Tables 1 to 3 give the average results for metric (1) to (3) as a function of node number. Clearly, the location update cost for our method is much lower than for HGM. This is mainly because the location update messages in our method could be sent in one packet. It is also clear that the cost of a specific location query in our method is much lower than for HGM. This advantage is the result of its properties: quick search within the adjacent low level squares and visiting higher level candidate location server first. However, for randomly selected location queries, the costs of the two compared methods are almost the same.

The reason is that the quick search within adjacent low level squares not always finds the desired location servers. In such cases the distance traveled by the quick search just increases the cost without any benefit. However, our method still benefits from visiting higher level candidate location servers first, preventing unnecessary travel by a location query packet. Thus the overall costs for these two methods are almost the same. Still, our method prevents the worst cases of location query travel such as shown in Fig. 3.
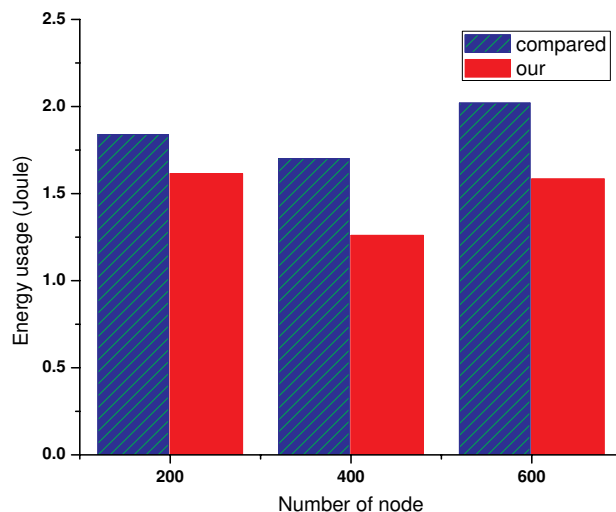


**Fig. 5.** Energy usage for static networks

The energy usage for both methods is shown in Fig. 5. As expected, the energy usage for our method is lower, in the range of 74% to 88% of the HGM energy use.

Table 4 shows the location query success rate for both methods. This rate is a little higher for our method.

**Table 4.** Static network results for metric (5)

| Node Number | 200 | 400 | 600 |
|---|---|---|---|
| Our method: | 99% | 96% | 96% |
| Compared method: | 91% | 94% | 93% |

*Mobile Network Scenario*

In the mobile network scenario, nodes, including location servers, move according to the random way-point 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. We did not run the specific location query simulation, because it is very difficult to find a source-destination pair when nodes are mobile. All the other configurations are the same as for the static network scenario. The $d(L_i)$ used in the lazy update procedure is set to $L_i/20i$ and the handover threshold is set to 90 m.

**Table 5.** Mobile network results for metric (1)

| V$_{max}$ (m/s) | 2.5 | 5 | 7.5 |
|---|---|---|---|
| Our method: distance | 324756 | 352415 | 388755 |
| (hops) | (3581.9) | (3903.1) | (4528.7) |
| Compared: distance | 475739 | 529158 | 575243 |
| (hops) | (4946.3) | (5736.5) | (6327.1) |

**Table 6.** Mobile network results for metric (2)

| V$_{max}$ (m/s) | 2.5 | 5 | 7.5 |
|---|---|---|---|
| Our method: distance | 1022 | 1018 | 967 |
| (hops) | (10.7) | (11.0) | (10.4) |
| Compared method: distance | 959 | 1055 | 1081 |
| (hops) | (10.1) | (11.3) | (11.7) |

Tables 5 and 6 show the average results for metric (1) and (2) as a function of speed $V_{max}$. The location update cost grows for both methods with the increase of speed $V_{max}$ but it is much lower for our method than for HGM for two reasons. First, our method sends location update in one packet. Second, we use the lazy update which reduces the update cost when node oscillates near the square boundary. For the randomly selected location queries, the costs for these two methods are almost the same.
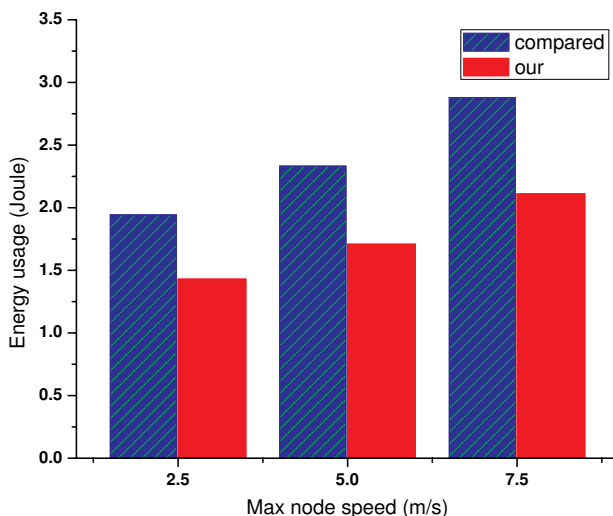


**Fig. 6.** Energy usage for mobile networks

The energy usage for mobile scenario is shown in Fig. 6. As expected, the energy usage grows with the increase of speed $V_{max}$. However, our method uses only 73% of the energy used by the HIGH-GRADE method.

**Table 7.** Mobile network results for metric (5)

| V$_{max}$ (m/s) | 2.5 | 5 | 7.5 |
|---|---|---|---|
| Our method: distance (hops) | 80% | 72% | 59% |
| Compared method: distance (hops) | 74% | 69% | 56% |

Table 7 shows the location query success rate for mobile scenario. The rate drops with the increase of $V_{max}$ but it is still a little higher for our method than for HGM.

## V. APPLICATION OF TECHNIQUE TO A GAIAN DATABASE

The Gaian Database [11] is a dynamic, distributed federated database which combines the principles of large distributed databases, database federation, and network topology in a dynamic, ad-hoc environment. The GaianDB uses a controlled flood query mechanism to locate the data from across the distributed database node. This mechanism has been shown to be highly scalable for simple queries in networks where the data is not partitioned, since the flood query is used to discovering the optimal paths for routing the discovered data back to the querying node [12]. The GaianDB implements a preferential attachment algorithm based on a 'fitness function' determined by each existing node in the database network. In one implementation of this approach a joining node is more likely to connect to a node which already has a larger number of connections. The aim of this approach is to reduce the diameter of the network. The number of hops between nodes is reduced to minimize the total time required to perform the broadcast-style queries of the prototype Gaian Database.

Whilst the mechanism is efficient for situations where the data is not partitioned, in many situations and particularly in a MANET, the data may be partitioned by the geographic location of the node at which the information is being stored. In this case, if a query for data uses the geographic location as a predicate then the query could be optimally routed just to those nodes that lie within the geographic location determined from the query predicate. In this case the query would be routed to the base location servers and then a localized flood query could be performed to all other nodes with one hop of the location server.

An alternative mechanism to the flood query is therefore to make use of server hierarchy defined in this paper for efficient broadcasting of a query and efficient gathering the responses in the query source. Clearly, using the location server hierarchy in broadcast implementation, we can ensure that a message will be received by each node once and only once in the broadcast procedure.

The resulting approach is similar to that proposed in [13] in the more general case of data partitioning using a Content Addressable Network (CAN). In the proposed mechanism we use the geographic location as the partitioning key for the data to enable at each server which specific regions covered by server contain nodes with the specific content within a network and to send a query to only these nodes of interest.

On further alternative, considered in [14] is to define a hypercube topology by assigning a hypercube label to each node. The imposition of a hypercube structure should provide advantages when routing queries to known destinations. Hypercube labeling also provides an efficient mechanism to broadcast messages, where we can guarantee that a message will

be received by each node once and only once in the broadcast procedure.

Maintaining a CAN embedded in the location servers and processing the queries using CAN will be subject of our future work. In the case of a MANET environment this approach leads to an interesting model for how nodes should exchange data as they move between regions and this is the subject of current investigations.

## VI. CONCLUSION

In this paper, we introduced a novel location service for mobile wireless networks that aims at reducing the overall energy cost by decreasing the distance traveled by the location update and query packets. Main contributions of the paper include: (i) novel location update packet routing, and (ii) novel location query packet routing that together reduce the average number of hops traversed by those packets, thereby reducing the average energy cost of our location service. Extensive simulations are performed to demonstrate that the new scheme achieves energy efficiency while maintaining all the other performance metrics. The approach can also be used to facilitate the efficient routing of queries in the Gaian Database [11] where the data is partitioned using a location as a partitioning key. In future work, we will use this location service in designing routing protocols and applications for mobile wireless networks including query execution on mobile networks. We will also be investigating the costs of exchanging data between mobile nodes in order to maintain the geographic partitioning of the data.

## ACKNOWLEDGEMENT

## REFERENCES

[1] Basagni, S., Chlamtac, I., Syrotiuk, V. R., Woodward, B. A.: A Distance Routing Effect Algorithm for Mobility (DREAM). In: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom), pp. 76--84. IEEE Press, New York (1998)

[2] Camp, T., Boleng, J., Wilcox, L.: Location Information Services in Mobile Ad Hoc Networks. In: Proceedings of the IEEE International Conference on Communications, pp. 3318--3324. IEEE Press, New York (2002)

[3] Liu, D., Stojmenovic, I. Jia, X. H.: A Scalable Quorum Based Location Service in Ad Hoc and Sensor Networks. In: Proceedings of the IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS), pp. 489--492. IEEE Press, New York (2006)

[4] Woo, S. C., Singh, S.: Scalable Routing Protocol for Ad Hoc Networks. ACM Wireless Networks, vol. 7(5), pp. 513--529 (2001)

[5] Das, S. M., Pucha, H., Hu, Y. C.: Performance Comparison of Scalable Location Services for Geographic Ad Hoc Routing. In: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1228--1239. IEEE Press, New York (2005)

[6] Yan, Y., Zhang, B. X., Mouftah, H.T., Ma, J.: Hierarchical Location Service for Large Scale Wireless Sensor Networks with Mobile Sinks. In: Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM), pp. 1222--1226. IEEE Press, New York (2007)

[7] Ahmed, S., Karmakar, G. C., Kamruzzaman, J.: Hierarchical Adaptive Location Service Protocol for Mobile Ad Hoc Network. In: Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), pp. 1--6. IEEE Press, New York (2009)

[8] Yu, Y. Z., Lu, G.-H., Zhang, Z.-L.: Enhancing Location Service Scalability with HIGH-GRADE. In: Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), pp. 164--173. IEEE Press, New York (2004)

[9] Abraham, I., Dolev, D., Malkhi, D.: LLS: a Locality Aware Location Service for Mobile Ad Hoc Networks. In: Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), pp. 75--84 (2004)

[10] Flury, R., Wattenhofer, R.: MLS: an Efficient Location Service for Mobile Ad Hoc Networks. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 226--237. ACM Press (2006)

[11] G Bent, P Dantressangle, D Vyvyan, A Mowshowitz, V Mitsou: A Dynamic Distributed Federated Database, In: Proceedings of the 2nd Annual Conference of International Technology Alliance (2008).

[12] G. Bent, P. Dantressangle, P. Stone, D. Vyvyan, A. Mowshowitz: Experimental Evaluation of the Performance and Scalability of a Dynamic Distributed Federated Database, In: Proceedings of the 3rd Annual Conference of International Technology Alliance (2009).

[13] P. Fraigniaud, P. Gauron, "D2B: a de Bruijn Based Content-Addressable Network", Theoretical Computer Science, vol. 355, no. 1, pp. 65--79 (2006).

[14] P. D Stone, P. Dantressangle, G. Bent, A. Mowshowitz, A. Toce, B. K. Szymanski: Relational Algebra Coarse Grained Query Cost Models for DDFDs, Submitted to 4th Annual Conference of International Technology Alliance (2009).