Ad Hoc Networks 11 (2013) 273–287

Contents lists available at SciVerse ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Energy-efficient location services for mobile ad hoc networks ☆

Zijian Wang [a,b,1], Eyuphan Bulut [a], Boleslaw K. Szymanski [a,*]

[a] Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
[b] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

ARTICLE INFO

Article history:
Received 5 January 2012
Received in revised form 24 May 2012
Accepted 29 May 2012
Available online 7 June 2012

Keywords:
Mobile ad hoc networks
Location service
Energy efficiency

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. Several location service schemes have been proposed, but the most promising among them, hierarchical hashing-based protocols, rely on intuitive design in the published solutions. In this paper, we provide full analysis of the efficiency of routing in hierarchical hashing-based protocols as a function of the placement of the routers. Based on the theoretical analysis of the gain and costs of the query and reply routing, we propose a novel location service protocol that optimizes the distance traveled by the location update and query packets and, thus, reduces the overall energy cost. These gains are further increased in the second presented protocol by the optimal location of servers that we established through analysis of geometrical relationships between nodes and location servers. Simulation results demonstrate that the proposed protocols achieve around 30–35% energy efficiency while improving or maintaining the query success rate in comparison to the previously proposed algorithms.

1. Introduction

Recently, there has been an increasing usage of mobile devices such as smartphones, iPads and GPS devices by people and vehicles. The applications running on these mobile devices require ad hoc type of communication, and therefore necessitated the design of new cost efficient routing algorithms for MANETs with constantly changing topology. Since these mobile devices are carried voluntarily by people (the power of these nodes is not consumed for mobility), the main factor that depletes the energy of such devices is the set up and maintenance cost of routing algorithms that provide the communication between the nodes [1].

Among many routing algorithms proposed for MANETs, location based routing has received much attention and is considered to be the most efficient and scalable routing paradigm [2]. 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. Since mobile nodes are battery powered with limited energy, energy efficiency must be taken into consideration when designing location service protocols.

1.1. Related work

There have been various protocols proposed for location service. The earliest of them were flooding-based ap-

☆ Initial version of this work is published in [26].

* Corresponding author. Address: Lally 204, 110 8th street, Troy, NY 12180-3590, USA. Tel.: +1 518 276 2714; fax:+1 518 276 4033.
E-mail addresses: wangz@cs.rpi.edu (Z. Wang), bulute@cs.rpi.edu (E. Bulut), szymansk@cs.rpi.edu (B.K. Szymanski).

[1] Most of the work was done when Zijian Wang was with Rensselaer Polytechnic Institute, Troy, NY, 12180. Currently, he is with the Institute of Computing Technology, Chinese Academy of Sciences.

1570-8705/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.adhoc.2012.05.012

proaches. DREAM [3], DLS, and SLS [4] 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 [4]) 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 [5], 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 [6] and in multi-directions [7].

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 [8,9], 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 [10] was proposed to fix some of the above drawbacks.

The notion of hierarchical structure used for location service was first introduced in [11]. In the hierarchical hashing-based protocols [12–14], 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. Although energy-related parameters are considered in some routing protocols such as [15], location service protocols mainly focus on the ability to find the location of the destination nodes. Thus, their designs are not supported by the rigorous analysis of the energy efficiency of forwarding location update and location query packets.

1.2. Contributions

In this paper, we focus on the rigorous analysis of the energy efficiency of the location update and query routing and its impact on the optimal design of location service protocols. We also present a novel protocol resulting from this analysis that optimizes energy consumption of the protocol induced communication. We focus our analysis on hierarchical hashing-based protocols introduced in [11–14]. They use the hierarchical grid with servers randomly assigned to each node based on its ID. In this paper, we made the following fundamental contributions to this approach:

1. We analyzed analytically the gain and cost of the location query routing and derived the optimal location query and update strategies.
2. Based on this analysis, we proposed an efficient protocol for reducing the distance traveled by location update and query packets. The proposed protocol decreases the energy cost of location service, increases the delivery ratio, and balances the location service load equally among all nodes.
3. We proposed a second algorithm that takes advantage of the existence of the optimal locations (that we identified) for the servers; assigning server duties to nodes near these optimal locations brings the energy consumption and the delivery ratio close to their optimal values.
4. We performed extensive simulations with many varying parameters running over many different environments to demonstrate experimentally the advantages of proposed protocols over the leading existing protocols.

The remainder of the paper is organized as follows. 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 protocols in detail. Section 4 provides the simulation results and compares our protocols with existing work. 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[2] square area. Each node has a unique ID, and is equipped with a communica-

[2] The protocols discussed here can easily be extended to three dimensional space. For simplicity, we used two dimension.

tion 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 [18,19]) 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. For a non square area, it could be covered by a square with minimum size. 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($1 < i \leqslant N$) 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 $1 \leqslant i \leqslant N$.

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

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

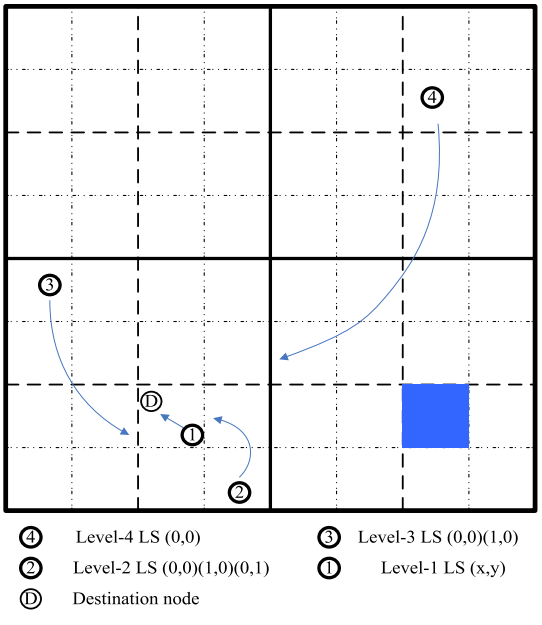


Fig. 1. An example for a 4-level hierarchy network.

where $\left(s_x^i, s_y^i\right)$ ($s_{x|y}^i$ 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:

$$s_{x|y}^i = \sum_{k=1}^{N-i} L_{N-k} \times a_{x|y}^{N-k} \tag{2}$$

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 $\left(na_x^{N-1}, na_y^{N-1}\right) \cdots \left(na_x^i, na_y^i\right)$ (in short $na_{x|y}^i$) of the level-i square to which this node belongs is calculated using the following formula:

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

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

This section first gives the details of how the location update procedure is performed to reduce the distance traveled by location update packets in Section 3.1. Then a novel location query method aiming to reduce the distance traveled by location query packets is proposed in Section 3.2. At last, another novel algorithm which optimizes the location server positions to reach the same aim is presented in Section 3.3.

3.1. Location update

The following key issues need to be addressed in any attempt to reduce the distance traveled by the location update packets: (1) which nodes are selected as location servers and the rules of updating location information on these location servers (Section 3.1.1); (2) how location information is organized and stored on each location server (Section 3.1.2); (3) how location information is handed over to new location servers when the old ones move out of position (Section 3.1.3); and (4) how location update packets are forwarded (Section 3.1.4).

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 $\left(ls_x^i, ls_y^i\right)$ (referred to as location server point) for each node in level-i square is determined as:

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

where ID is the unique identifier of the node and $\left(s_x^i, s_y^i\right)$ 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.[3] There may be no node at the exact location server point. In such a case, we choose the node nearest to the location server point as the corresponding location server using the perimeter based scheme presented in [14]. In perimeter based scheme, the location update packet sending to the location server point (LSP) will circulate on the perimeter around the LSP and the location information will be stored on all the nodes on the perimeter enclosing the LSP. This scheme will guarantee that there will be at least one location server around the LSP, even when the node distribution is non-uniform across the network area and there are holes in the network.

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 ($i \geqslant 1$) square S_i^{old} into new level-i square S_i^{new} within the lowest level-k ($k \geqslant i+1$) 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 ($1 \leqslant j \leqslant i$) location servers in S_i^{new}. Third, it sends location remove packets to all of its outdated level-j ($1 \leqslant j \leqslant i$) location servers in S_i^{old}. 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.

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. Lazy update allows a node to move out of level-i square up to a certain distance without updating corresponding location servers. This scheme will keep the location query to be efficient and locality aware, and reduce the overhead due to oscillating nodes, as verified in [16,17]. In our case, we 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 storage

In the proposed method, each location server node maintains a list of IDs of those nodes for which location information is stored on this server. 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).

[3] In the simulations, we used the following simple hash function. For level-k ($1 \leqslant k \leqslant N$ for ADJ and HG methods and $k = 1$ for OPT method) 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.

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[4] in the list. That is, for the example in Fig. 1, where $N = 4$, the memory usage is only 340 bits[5] 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.

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. Here, 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.

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 node 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. When receiving a location handover packet, node will check whether its ID is in the list of location servers to be informed. 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.

[4] 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.

[5] 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.

3.1.4. 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 (referred to as update node here after) 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 to as d-indiv) and (ii) when they are sent in one packet that traverses all the desired location servers (referred to 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 update node (64 bits). All the other table entries store only location server level ($\lceil \log_2 i \rceil$ bits) and the address sequence for the level-$(i-1)$ square in which the update node resides ($2(N-i+1)$ 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 update 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 $O(N^2)(O(\log(n)^2))$.

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

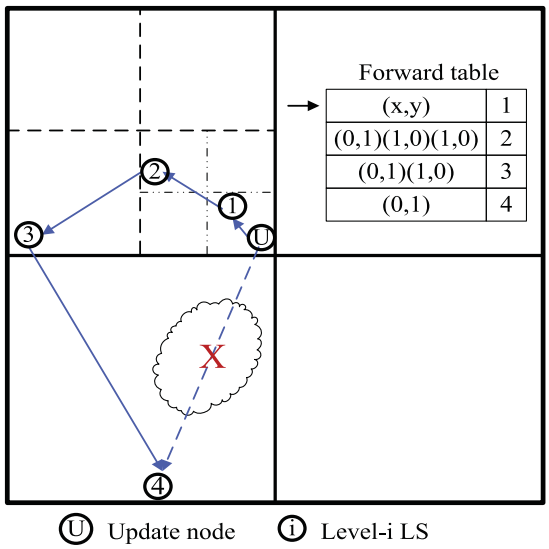


Fig. 2. An example of location update and forwarding table.

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. Proposed location query method

Here, we focus on the most important step, location query processing, that is used to find the proper location servers to obtain location information. We first introduce the observation that inspires our new location query method. Then we analyze the gain and cost of using this new method and introduce the location query procedure in detail.

3.2.1. Observations and basic idea

We made the following observations about the previous methods described in [14–16]. In these methods (throughout the paper we specifically refer to HIGH-GRADE method in [14], abbreviated as HG here, as the representative of such methods), 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 the corresponding square 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 right 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[7] in Fig. 3. The destination node and its level-1 location server are

[7] Note that in Figs. 1–4, for illustration purposes, the locations of LS nodes are computed randomly (without using the hash function we used in our simulations) because various hash functions can be utilized in different hierarchy based algorithms.

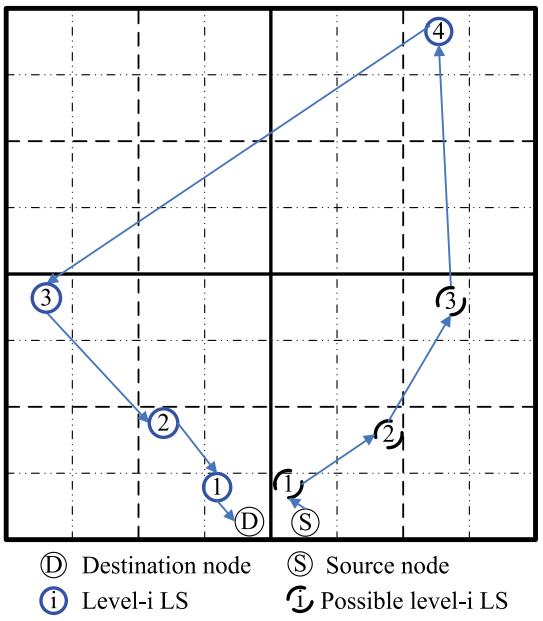


Fig. 3. Location query scheme from [14].

within adjacent square of the source node, but the location query packet has to visit level-1 to level-3 possible location servers and level-4 to level-1 real location servers to find the destination node. 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 would decrease the distance traveled by packet. In fact, if the high level candidate location server is not a right 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.

Some schemes (such as [16,17]) try 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. Even though this helps in finding the nearby location servers quickly, the location query packet still travels a long distance if the location servers are far away from the source.

Considering the above points, we conclude 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 the conclusions of our observations above, 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[8] and then introduce the location query algorithm step by step in detail.

For simplicity, we call the possible location servers within source node's level-k square as base location servers, and denote level-k base location server point as LSP_k. We call the possible location server in adjacent level-k square of source node as extra location servers, and denote level-k extra location server point as LSP_{k_a}.

3.2.2.1. 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 extra location servers, then we gain by avoiding sending first a query packet to a sequence of base location 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: Input: source node's position and destination node ID
2: Output: forwarding table of the location query packet
3: Determine which method to use (ADJ or HG)
4: Find optimal visiting path for base LSPs (Algorithm 2)
5: if ADJ is used then
6:    Find adjacent squares to be searched
7:    Compute gain and cost for each adjacent square found
8:    if gain > cost then
9:       Put corresponding extra LSP into visiting path
10:   end if
11:   Sort all the LSPs in visiting path using Hamiltonian path method
12: end if
13: Generate forwarding table based on sorted visiting path
```

If the source node visits one level-k extra location server 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 base location servers using HG method. In this case, we get no gain but pay the extra cost. Assume LSP_1 is the first base location server point that the source node will visit using HG method. Then the cost will be:

[8] A detailed analysis and proof procedure can be found in [22].

$$c_k = (4^{-N+k})d(source, LSP_{k_a}) + (1 - 4^{-N+k}) \times [d(source, LSP_{k_a}) + d(LSP_{k_a}, LSP_1) - d(source, LSP_1)] = d(source, LSP_{k_a}) + (1 - 4^{-N+k})[d(LSP_{k_a}, LSP_1) - d(source, LSP_1)] \quad (6)$$

3.2.2.2. Determining which method to use. The first step is to determine which method to use in the location query procedure, ADJ or HG. 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 meaning behind this scheme is that ADJ method will be selected only if the estimated maximum gain is big enough to pay the extra cost of visiting adjacent squares. 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_1s to LSP_2 (referred to as $L(1,2)_{max}$). Thus, Eq. (5) becomes:

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

It is easy to prove [22] that g_k is a strictly increasing function of k, thus we have the maximum g_k when $k = N - 1$:

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

3.2.2.3. Finding optimal visiting path for base LS. The second step is to find the optimal visiting sequence (path) for base location servers points. In original HG method, the source node visits base location 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 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 (the path is 0x11), 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 (the path is 0x10), 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 sequence finding process is shown in Algorithms 2 and 3.

Algorithm 2. Optimal visiting sequence for base LS

```
1: Input: position list of original visiting base LSP
2: Output: position list of optimized visiting base LSP
3: opt_path = 2^(N-1) − 1
4: opt_dist = Length(opt_path)
5: for i = 0; i < 2^(N-1) − 1; i++ do
6:    dist = Length(i)
7:    if dist < opt_dist then
8:       opt_dist = dist
9:       opt_path = i
10:   end if
11: end for
```

3.2.2.4. Finding adjacent squares to be searched. If ADJ method is used, the source node has to determine which adjacent squares will be searched for extra location servers. In order to narrow down the possible adjacent squares to be searched, we use the following process. After finding the optimal visiting path of base location servers, 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 $r_{est} = [L(s,N) + L(1,2)_{max}\left(\sum_{i=k}^{N-1} 2^{i-1}\right)]4^{-N+k}$. If this circle intersects with level-k ($1 \leqslant k \leqslant h$) squares contained within another level-h square (not the one that contains the source node), the source node will consider the corresponding level-k squares as candidate adjacent square. For each of the level-k candidate adjacent square, 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 dropped.

Algorithm 3. Length (int path)

```
1: Input: the visiting sequence (path) of base LSP
2: Output: the length of the input visiting path
3: dist = 0
4: last_node = 0
5: for i = 1; i ⩽ N; i++
6:    if ((i^th digit of path from right is 1) || (i = N)) then
7:       if (last_node=0) then
8:          dist = dist + d(LSP_last_node, LSP_i)
9:       else
10:         dist = dist + d(LSP_last_node, LSP_i) ×
          (1 − 4^(−N+last_node))
11:      end if
12:      last_node = i
13:   else
14:      dist = dist + d(LSP_i, LSP_i+1) × 4^(−N+i)
15:   end if
16: end for
17: return dist
```

3.2.2.5. Determining visiting path of all LS. For each candidate adjacent square remained after the previous step, the source node will compute the exact gain and cost using Eqs. (5) and (6). If the gain exceeds the cost, the corresponding extra location server point in the candidate adjacent square will be put into visiting path. All the location servers (including base and extra location server) in the visiting path will be sorted using Hamiltonian path method.

At last, forwarding table of location query packet is generated based on the sorted visiting path.

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 optimizing the visiting path of four base location server points, the level-1 base 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 path. Then, all location server points in the visiting path are sorted and the forwarding table is generated as shown in Fig. 4.

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 $O(N)$ ($O(\log(n))$).

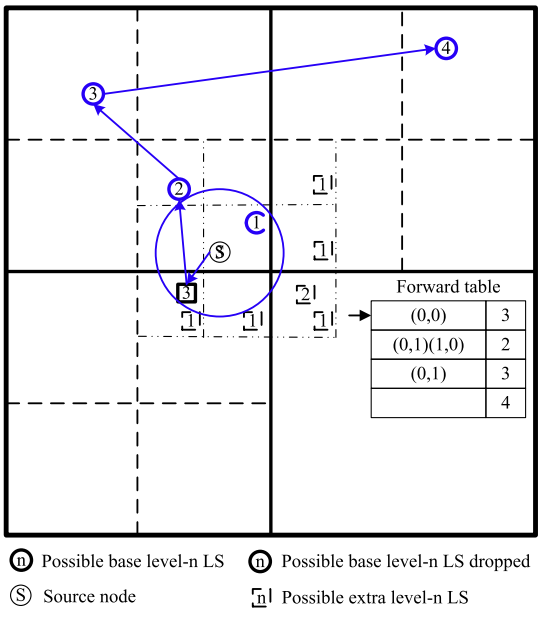


Fig. 4. Location query procedure and forwarding table.

3.3. Location server position optimization

In the previous section, we tried to reduce the location 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. 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. 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 $a + b$.

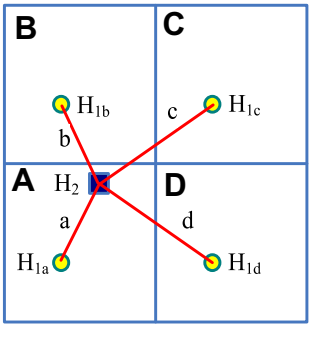


Fig. 5. A two-level grid example.

By enumerating all possibilities, it is easy to show that the average cost of query location is $\frac{3(a+b+c+d)}{8}$. The sum $a+b+c+d$ is minimized when the server is located at the center of the square (elementary but nice proof of this fact is presented in [22] but omitted here for the sake of brevity).

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 location server (x_i,y_i) could be computed as:

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

Although the distribution of location servers are not even using the above equation, the energy cost is still evenly distributed over the whole network as shown in the simulation section.

We can also estimate the benefit from the optimization of location server positions. When the locations of servers are determined by Eq. (4), we can compute the average location query cost for non-optimal placement as follows. Given one of the possible level-1 location server points, (x,y), the other three level-1 location server points will be $(x+L_1,y)$, $(x,y+L_1)$, $(x+L_1,y+L_1)$, where L_1 is the side length of level-1 square. And the level-2 location server point will be $(2x,2y)$. Assuming a uniform distribution of the location server within one level-1 square, the distance $(a+b+c+d)$ as shown in Fig. 5 can be computed as

$$\int_0^{L_1}\int_0^{L_1} \sqrt{x^2+y^2} + \sqrt{(L_1-x)^2+y^2} + \sqrt{x^2+(L_1-y)^2} + \sqrt{(L_1-x)^2+(L_1-y)^2}\,dx\,dy \qquad (8)$$

By using Matlab integration tools, the result of Eq. (8) is $3.0608L_1$, and the average distance yielded is $1.1478L_1$. When the location servers are placed at the optimal positions, then $a=b=c=d=\sqrt{2}L_1/2$, thus the average cost of location query for optimal placement will be $1.0607L_1$. Above all, 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 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 [14] (referred to as HG) and the method presented in [13] (referred to as SALS).[9] 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, and it is partitioned into 4-level squares for HG, ADJ and OPT as shown in Fig. 1. For SALS, the network is partitioned into 3-level squares, 5×5 level-1 squares with side length of 100 m create a level-2 square, and 2×2 level-2 squares create one level-3 square. We use this different 3-level configuration here because the simulation code of SALS we get from the authors of [13] is written under this configuration. In order to compare the results of SALS with other three methods fairly, we analyzed the theoretical performance of SALS for 3-level configuration (referred to as SALS3) and 4-level configuration (referred to as SALS4) in appendix, and estimated the performance of SALS4 from the simulation results of SALS3.

IEEE 802.11 is used as the MAC and physical layer protocol. We used two-ray-ground propagation model. Each node's transmission range can vary 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} can be found in NS2 documentations [20].

4.2. Performance metrics

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

1. The average total distance[10] 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[11] 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*.

[9] We would like to thank the authors of [13] for kindly providing the simulation code of their method.

[10] 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.

[11] Since all the methods compared use hello messages with the same interval and packet size, the cost of those messages is not included in total energy usage. Besides, all the methods ran for the same time duration, all nodes almost have the same time in idle state, which leads to same energy cost of idle state, thus it is not included in total energy usage neither.

4.3. Simulation results

This section presents the results of our evaluations of the four mentioned algorithms according to the aforementioned metrics.

We randomly generated five topologies for each configuration. For each of them we ran 20 groups of simulations. Each simulation ran for $S + 45$ s, where S is the number of nodes. For the first S seconds, node s sent location update at sth second.[12] During this procedure, the location servers (including both optimal and non-optimal) placement can be determined. All the nodes start to move after $(S + 20)$th second, according to the random way-point[13] mobility model with no pause time. New nodes near the location server points will be assigned as new location servers by the "handover" procedure. 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 1 m/s to 2 m/s. These are the moving speeds of walking people or robots. 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 s interval between its queries. 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.[14]

For OPT method, it should be noted that the optimal placement is used to select the initial assignment of server duties to nodes at the beginning of the simulation. Then, all nodes move according to the random way-point model and some original server nodes may pass their duties to the nodes nearer than them to the optimal points through the "Handover" procedure.

Fig. 6 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 and SALS 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. 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 10.8%, which is very close to the analysis result 8.2% in Section 3.3, the better performance than estimation results from node mobility. 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)

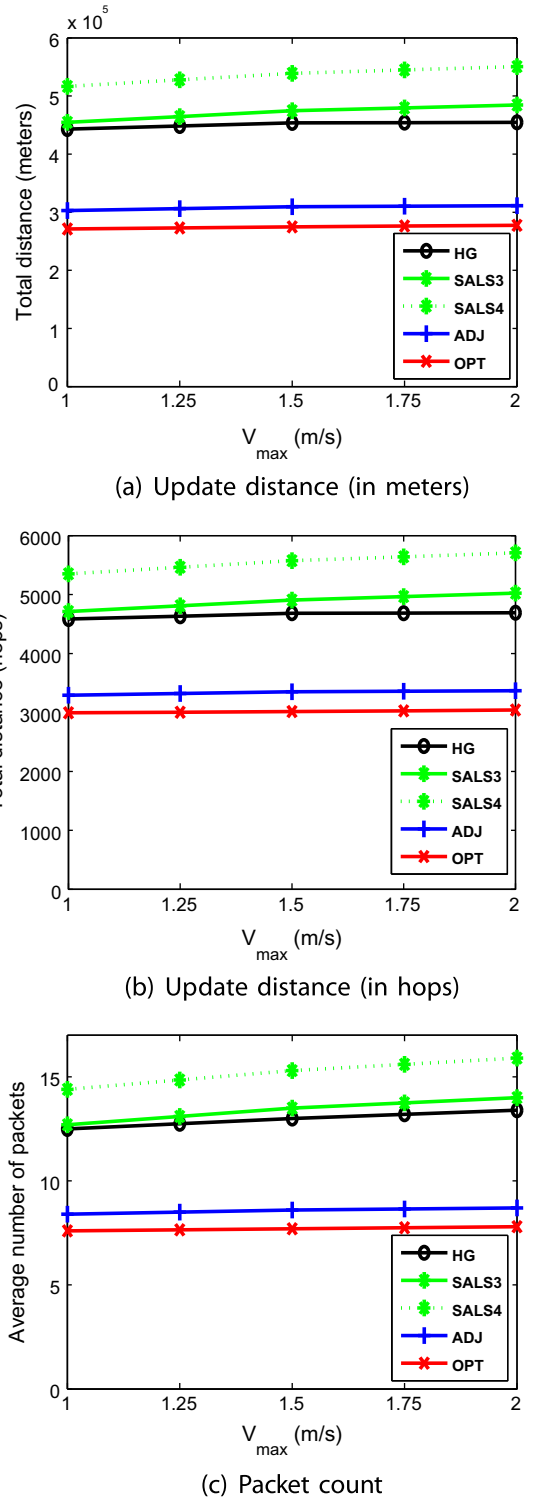
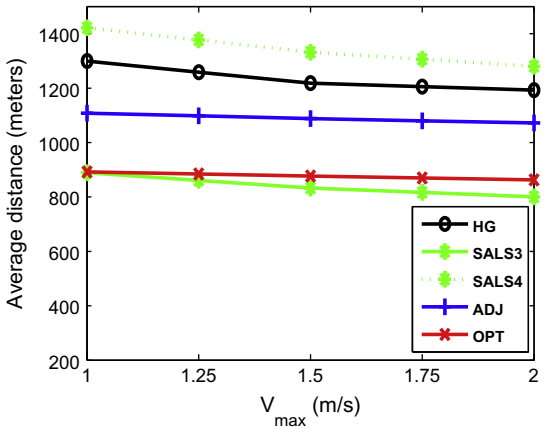


Fig. 6. Performance comparison of four algorithms in terms of location update distance and packet count.

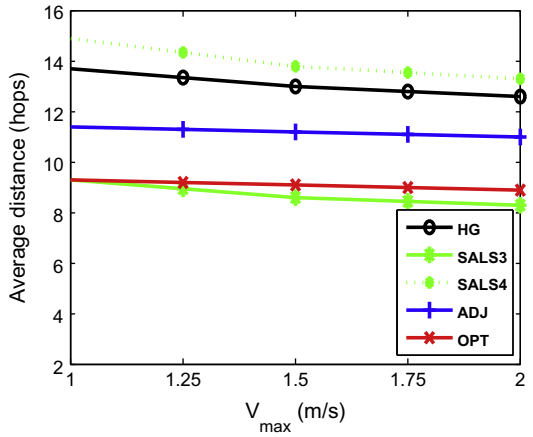
[12] Here, we applied this scheme to avoid packet collision in the initial location update procedure. When nodes start to move, they will send location updates only when necessary.

[13] This model is also used in related previous work [13,14]. Thus, we used it for fair comparison to previous work.

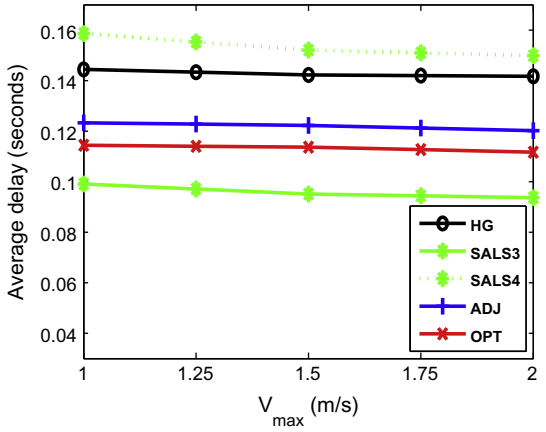
[14] We chose these values after an extensive run of simulations with different values.



(a) Query distance (in meters)



(b) Query distance(in hops)



(c) Query delay

Fig. 7. Performance comparison of four algorithms in terms of location query distance and location query delay.

shown in Fig. 6c look similar to the ones in Fig. 6a and b, which also verifies the simulation results in these figures.

Fig. 7 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, with an average improvement of 11.8%. 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 require that there is an upper limit for the location query delay. Moreover, the cost of OPT method is much lower than ADJ and HG because the optimized positions of location servers provide additional cost savings.

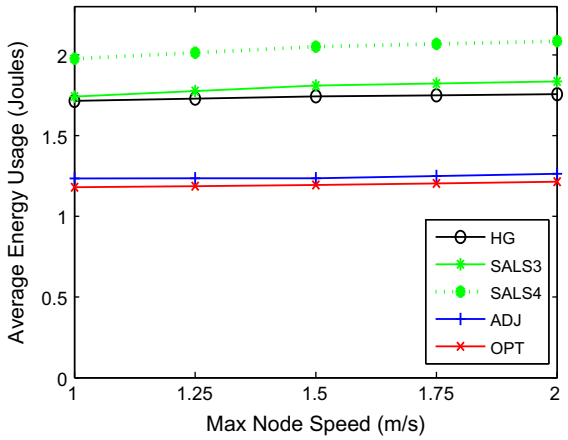


Fig. 8. Average energy usage per node.

Table 1
Average location server count.

V_{max} (m/s)	1	1.5	2
HG	278	283	289
SALS3	37	38	41
SALS4	37	38	41
ADJ	277	282	289
OPT	174	176	182

The performance of SALS3 is almost the same as OPT. Here, note that SALS3 uses 3-level structure, thus a lot of query packets will find desired information within level-2 square, with higher location update cost, as shown in Fig. 6a. When the simulation results of SALS3 are mapped to SALS4, its performance is even worse than HG.

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 and SALS 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, in addition to the reason discussed for HG and SALS method, the chance of improvement by 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

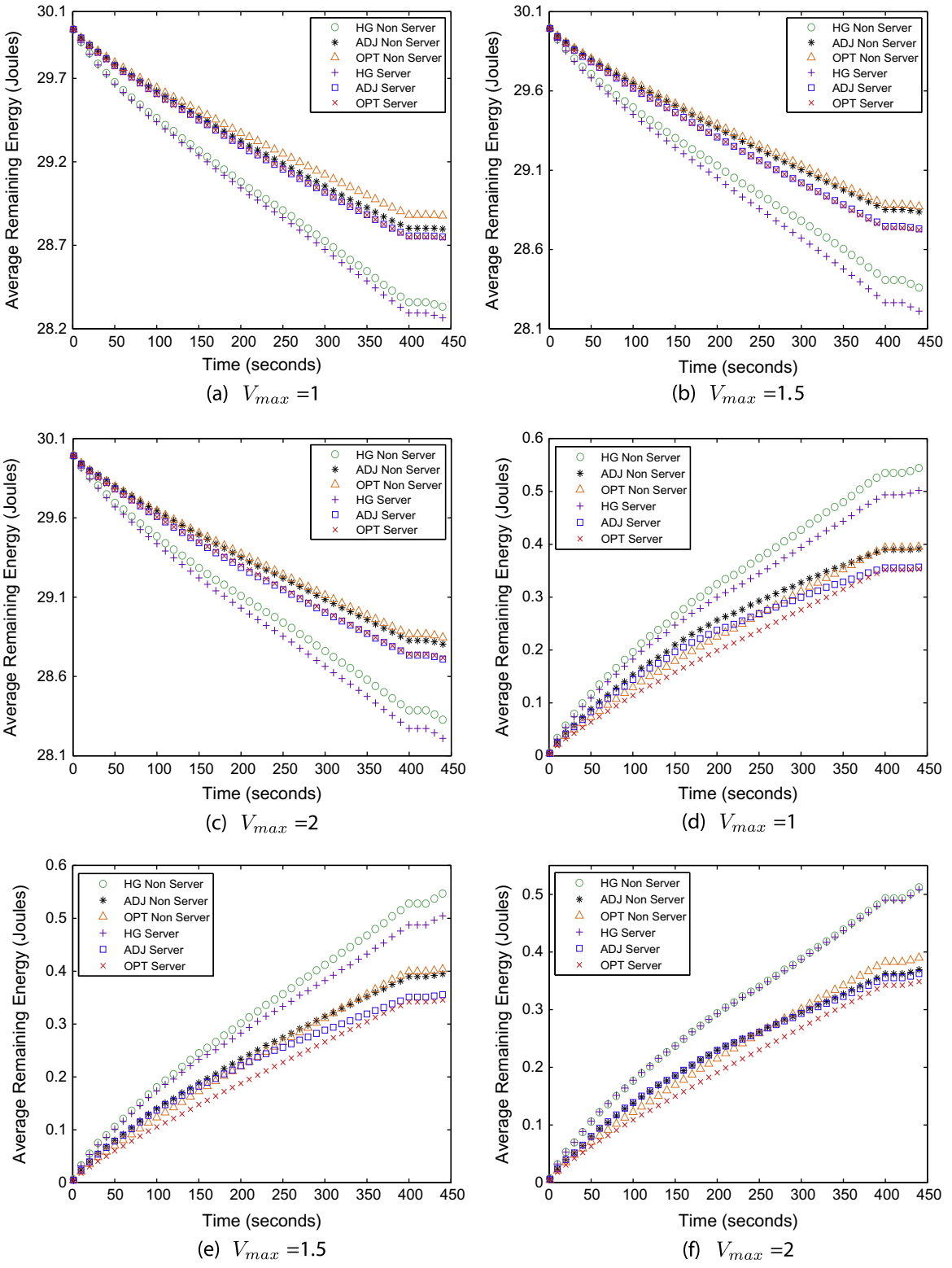


Fig. 9. Average remaining energy and variance.

Table 2
Average location query success rate.

V_{max} (m/s)	1 (%)	1.5 (%)	2 (%)
HG	83.8	78.2	79.8
SALS3	90.1	90.4	91.6
ADJ	89.5	84.9	83.2
OPT	91.0	87.1	88.2

servers directly, which reduces the distance traveled by these packets.

Since the delay of location query packets is directly proportional to the hop distance of location query packets, the relation between the slopes of different algorithms shown in Fig. 7c is very similar to the slopes of results in Fig. 7a and b, which also verifies the simulation results in these figures. Here, even though SALS3 has slightly shorter delay (due to the usage of more update packets yielding more energy consumption, as shown Fig. 8) than our algorithms have, SALS4 incurs much longer delay when showing its predicted performance in the same environment.

The energy usage of algorithms is shown in Fig. 8. As expected, the energy usage grows (slightly) 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 69% of the energy used by the HG method, while OPT method uses even less of it. SALS3 and SALS4 use more energy than HG, which can be deduced from average packet count forwarded by each node as shown in Fig. 6c.

Table 1 shows the average location server counts used in each method (we still count a node as location server after it sends location handover packet and transfers server duty to other nodes). We can see that ADJ has almost the same location server number as HG, but OPT has much fewer location servers. However SALS3 and SALS4 use home region not hash functions to determine which nodes work as location servers, thus there are fewer such location servers than in case of other three methods.

In Fig. 9, 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 HG, ADJ and OPT methods. Because location update cost is much larger than location query cost, we only consider location update energy usage here. For SALS method, we could not record the energy log file due to its huge memory requirement and file size. As shown in Fig. 9, there are three distinct phases of energy usage by the network resulting from the way that the simulations are organized. First, from 0 to 400 s, all nodes send and forward location update packets. Then, from 400 to 420 s, there is a gap in node's activities before they start to move. Finally, after 420 s, as the result of movements, some of the nodes start to send location update, location query and handover messages, according to the rules introduced in the paper. 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 scattered 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. Additionally, 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 [21].

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.[15] 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 38% less than the location server count 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.

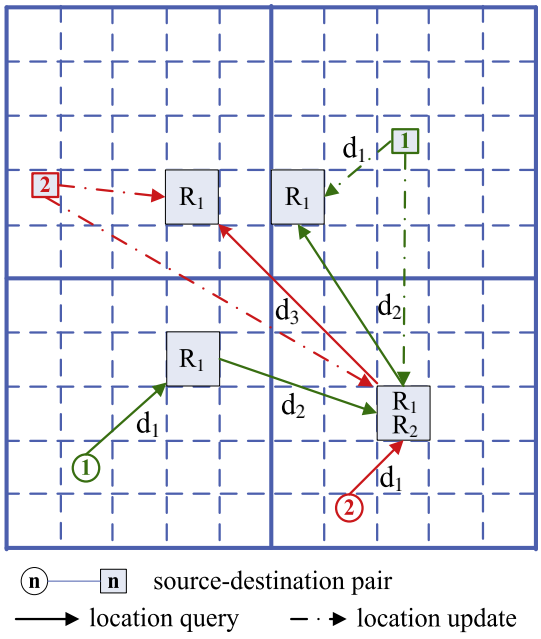


Fig. A.10. Location update and query in SALS.

Table A.3
Location query cost for SALS.

dst position	chance	query cost
L_{1-0}	1/4	d_1
L_{1-1}	1/4	$d_1 + d_2$
L_{1-2}	1/4	$d_1 + d_2 + d_3$
L_{1-3}	1/4	$d_1 + d_2 + d_2$

[15] 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 [22].

For SALS method, there are such few location servers and all the location servers are within one level-1 square, which will cause the energy usage hot spot problem.

Table 2 shows the average location query success rate for the compared algorithms. Most of them drop with the increase of V_{max} on average but still OPT has better success ratio than ADJ, while ADJ is better than HG. From Fig. 6c, we see that in HG method each node forwards more packets than ADJ and OPT methods. This increases the chance of packet collision that may result in location update or query packet loss, thus a decrease in success rate. SALS3 method has a little better performance than OPT method because of two reasons: (1) its location server region shift scheme will keep the location service stable when nodes are mobile; (2) it stores location information at each location servers within the level-1 square. Yet this little advantage is gained by increasing the cost of location update and storage.

5. Conclusion

In this paper, we present an analytical model for the performance of hierarchical hashing-based location server protocols. Based on this analysis, we introduced two novel location service protocols that optimize the overall energy cost of location service by decreasing the distance traveled by the location update and query packets. The first presented protocol, 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 overall performance 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 [23,24] for delay tolerant networks where the intermittently occurring contacts between nodes and low node density make 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 [25].

Acknowledgements

Research of Zijian Wang was supported by the China Scholarship Council and by the Center for Network Science and Engineering at RPI during his stay at RPI and by the National Basic Research Program of China (973 Program) under Grant No. 2011CB302803, and by the National Natural Science Foundation of China (NSFC) under Grant No. 61100179, and by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06030700. Research of Eyuphan Bulut and Boleslaw K. Szymanski 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 US 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.

Appendix A. SALS performance analysis

Fig. A.10 shows the location update and query in SALS under 3-level-configuration. Detail of these procedures can be found in [13]. The area is partitioned into 4 level-2 squares (referred to as L_{1-0} to L_{1-3} from lower left to upper right) and each of them is further divided into 25 level-1 squares. For each level-2 square, the level-1 square with most nodes in it serves as location server region (referred to as R_1 in Fig. A.10), and one of the four level-1 server regions serves as level-2 server region (referred to as R_2).

The distance (referred to as d_1 in Fig. A.10) traveled from one random selected node within L_1 to the corresponding R_1 equals to the average distance between two random points within L_1. Similarly, the distance traveled from one R_1 to the R_1 in the adjacent L_1 equals to the average distance between two random points within two adjacent L_1 (referred to as d_2 for two L_1s with same side, referred to as d_3 for two L_1s in diagonal positions, as shown in Fig. A.10).

For a random query, there is 1/4 chance that the source node is in L_{1-0}, and in such a case, the query cost is shown in Table A.3. Similarly, we can get the query cost when the source node is in other L_1s and finally we can get the average location query cost which is $d_1 + 3d_2/4 + 3d_3/8$. We can also get the average location update cost which is $d_1 + d_2/2 + d_3/4$.

Given the side length of the whole area, L, we can get the following results from numerical integration: $d_1 = 0.5214 * L/2$, $d_2 = 1.088 * L/2$,$d_3 = 1.4736 * L/2$. Thus, the location update and query cost are $0.7169L$ and $0.945L$, respectively.

Using the similar procedure, we can get the average location update and query cost for level-4 configuration, which are $0.8147L$ and $1.5112L$ respectively. Then, the performance of SALS4 can be estimated by $SALS4_{sim} = SALS3_{sim} * SALS4_{ana}/SALS3_{ana}$, where $SALS3_{ana}$ and $SALS4_{ana}$ are the analysis results for SALS3 and SALS4 and $SALS3_{sim}$ is the simulation results for SALS3.

The area of location server regions for SALS3 is $4*(L/2/5)^2 = L^2/25$, while for SALS4 it is $16*(L/4/5)^2 = L^2/25$. Thus we can see that they are the same, which results in the same number of location server nodes, as shown in Table 1.

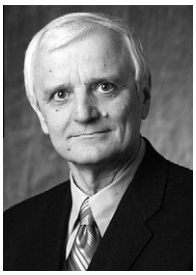
References

[1] J. Zhu, X. Wang, Model and protocol for energy-efficient routing over mobile ad hoc networks, IEEE Transactions on Mobile Computing 10 (11) (2011) 1546–1557.

[2] R. Friedman, 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.
[3] S. Basagni, I. Chlamtac, V.R. Syrotiuk, 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.
[4] 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.
[5] D. Liu, I. Stojmenovic, 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.
[6] F. Yu, Y. Choi, S. Park, E. Lee, M.S. Jin, 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.
[7] H. Jeon, K. Park, D.-J. Hwang, H. Choo, Sink-oriented dynamic location service protocol for mobile sinks with an energy efficient grid-based approach, Sensors 9 (3) (2009) 1433–1453.
[8] S.C. Woo, S. Singh, Scalable routing protocol for ad hoc networks, ACM Wireless Networks 7 (5) (2001) 513–529.
[9] S.M. Das, H. Pucha, 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.
[10] 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), 2005, pp. 258–263.
[11] J.Y. Li, J. Jannotti, D.S.J. De Couto, D.R. Karger, R. Morris, A scalable location service for geographic ad hoc routing, in: Proceedings of ACM International Conference on Mobile computing and Networking (MobiCom), 2000, pp. 120–130.
[12] Y. Yan, B.X. Zhang, H.T. Mouftah, 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.
[13] S. Ahmed, G.C. Karmakar, 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.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] A. Savvides, C.-C Han, 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), 2001, pp. 166–179.
[19] D. Niculescu, B.R. Badrinath, Ad Hoc Positioning System (APS) Using AOA, in: Proceedings of INFOCOM, 2003.
[20] http://www.isi.edu/nsnam/ns/.
[21] M. Grossglauser, D.N.C. Tse, Mobility increases the capacity of ad-hoc wireless networks, in Proceedings of IEEE Infocom, 2001.
[22] Z. Wang, E. Bulut, B.K. Szymanski, Energy-Efficient Location Service Protocols for Mobile Ad Hoc Networks, Technical Report 11-1, Department of Computer Science, RPI, 2011.
[23] E. Bulut, Z. Wang, B.K. Szymanski, Cost effective multi-period spraying for routing in delay tolerant networks, in: IEEE/ACM Transactions on Networking, vol. 18, 2010.
[24] E. Bulut, S. Geyik, 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), November 2010.
[25] S. Ahmed, G.C. Karmakar, 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), 2007, pp. 1577–1580.
[26] Z. Wang, E. Bulut, B. Szymanski, An energy efficient location service for mobile ad hoc networks, in Proceedings of 25th International Symposium on Computer and Information Sciences (ISCIS), September 2010.

Zijian Wang received his B.S. and Ph.D degree in electronics and information engineering from Beihang University, Beijing, China, in 2003 and 2010. From 2007 to 2009, he was a visiting Ph.D. student at Department of Computer Science, Rensselaer Polytechnic Institute (RPI), Troy, NY, USA. Currently, he is with Institute of Computing Technology, Chinese Academy of Sciences. His research interests are wireless ad hoc/sensor networking energy efficient protocols, including topology control, multi-path routing, target tracking, location service and service discovery.

Eyuphan Bulut (M'08) received his Ph.D. degree in computer science department of Rensselaer Polytechnic Institute (RPI) in 2011. He also holds B.S. and M.S. degrees from computer engineering department of Bilkent University, Ankara, Turkey. He is now with Mobile Internet Technology Group of Cisco Systems in Richardson, TX. His interests include design of protocols for wireless sensor and mobile ad hoc networks such as routing protocols for delay-tolerant networks.



Boleslaw K. Szymanski (M'82 F'99) is the Claire and Roland Schmitt Distinguished Professor of Computer Science and the Director of the Social Cognitive Academic Research Center led by RPI. He received his Ph.D. in Computer Science from National Academy of Sciences in Warsaw, Poland, in 1976. He is an author and co-author of over three hun- dred publications and an editor of five books. He is also an Editor-in-Chief of Scientific Programming. He is an IEEE Fellow and a member of the ACM for which he was a National Lecturer. His interests focus on parallel and distributed computing and networking.