

# Sharing Location Dependent Experiences in MANET

Kihwan Kim    Ying Cai    Wallapak Tavanapong

Department of Computer Science

Iowa State University

Ames, IA 50011-1040, USA

Email: {khkim,yingcai,tavanapo}@cs.iastate.edu

**Abstract**—This paper investigates a new problem of sharing location dependent experiences among mobile hosts in mobile ad-hoc networks. An experience is location and observer dependent. In other words, experiences of different people witnessing the same event may be quite different. The ability to retrieve prior experiences observed in a given area in advance is very useful for newcomers wishing to enter the same vicinity. Solving this problem is vital for important applications such as hurricane rescue missions, combat missions, and deep space or deep sea exploration. In this paper, we propose a distributed solution that lets mobile hosts share their experiences efficiently. Our simulation results show that our approach significantly outperforms a centralized approach.

## I. INTRODUCTION

Mobile Ad-hoc Network (MANET) has received tremendous research interests in recent years. MANET consists of a number of mobile hosts that communicate with each other through wireless packet relaying. MANET is excellent for sharing information in areas lacking of communication infrastructures such as disaster areas, combat zones, or unexplored territories (e.g., deep space or deep sea). In these scenarios, mobile hosts scatter around to observe various parts of a large unfamiliar territory and record their experience in their local storage. An experience is location and observer dependent data that may be in the form of text, images, audio, or videos. Each experience is associated with a recording time, location, and information about the host that creates the experience. Prior to moving to any new location, a mobile host queries for experiences observed by other mobile hosts in the close vicinity of the new location. The user of the mobile host review these experiences to determine her course of actions, depending on a specific application. For example, in a hurricane response mission, an emergency response personnel may avoid a location that was observed to be dangerous by another personnel. On the other hand, for space exploration, an astronaut may decide to move forward to examine the new location that has not been explored by others. To the best of our knowledge, efficient sharing of location-dependent experiences among mobile hosts in MANET has not been investigated in the literature and is the subject of our investigation in this paper.

A simple solution is a centralized approach that lets each mobile host report its location-dependent experience to a central server at a fixed location via multi-hop relays. The

server keeps past experiences of all the mobile hosts. Before moving to a new location, a mobile host asks the server for other hosts' experiences in the close vicinity of the new location. The centralized approach is not scalable since it requires frequent interactions between the server and each mobile host, especially when a large number of mobile hosts create many experiences. Furthermore, the mobile hosts exhaust their battery power quickly as they need to frequently relay experiences to/from the central server. Last, the server is a single point of failure.

To address these problems, an adhoc network of a number of stationary servers can be used. These servers have a long transmission range to communicate among themselves and are more powerful than a regular mobile host. Each mobile host reports its experiences to its nearest stationary server. These stationary servers communicate among themselves to relay relevant experiences to a querying host. This approach, however, requires additional servers, which may not be feasible or too costly for some applications.

A simple distributed approach is to treat a location-dependent experience as general data. Each host maintains its own experience. Before moving to a new location, a host broadcasts a request with the intended location information to its neighbors that relay the query to all the hosts through flooding. Only the hosts who have created experiences in the desired vicinity reply to the querying host. This approach is very expensive since a network-wide broadcast is needed every time a host moves.

In this paper, we propose a new, efficient distributed solution, taking advantage of location information and the following ideas to reduce communication overhead and provide fast response time.

- First, we separate a host's experience into an *experience summary* and a *data item*. The summary provides information such as the location and the time the experience is observed. The experience summary is typically very small. The corresponding data item can be large as it may be a video recording of an event. This separation enables us to employ different replication policies for data items and experience summaries. We store only one copy of a data item in the aggregated space of all mobile hosts to effectively utilize the storage space. We keep one or more copies of an experience summary in different

mobile hosts to provide a fast response time.

- Second and more importantly, we try to keep the experience summary and the corresponding data item in a mobile host close to the location where the experience was recorded. This host can provide any other host that wants to move into that location the relevant experiences quickly and efficiently. Our strategy takes into account of the different replication strategies for the summary and the data item.

Our contributions are as follows. We are the first to investigate the problem of sharing location dependent experiences in MANET. We propose a novel distributed solution that is shown to significantly outperform the approaches using a small number of stationary servers in our study.

The remainder of this paper is organized as follows. Section 2 discusses related work and reasons they are not efficient in solving the problem of sharing location dependent experiences. Section 3 presents the proposed algorithm and Section 4 reports our performance evaluation. Finally, we offer our concluding remarks in Section 5.

## II. RELATED WORK

A large number of research efforts in various aspects of MANET have been made in recent years, ranging from flooding and routing [1], data lookup and delivery [2], [3], [4], [5], [6], and service discovery [2], just to name a few. However, there are no prior work proposed to solve the problem of sharing location dependent experiences in MANET.

Several indexing techniques for spatial and temporal information of moving objects have been proposed [7], [8]. Such indexes can be employed in a stationary server to reduce the server CPU and disk I/O time spent to search for experiences in a query region. However, these indexes cannot reduce the more costly communication overhead between the server and the mobile hosts. Several efforts have also focused on designing efficient techniques to continuously monitoring mobile hosts moving in and out of a fixed monitored region [9], [10]. The monitored regions are distributed to all mobile hosts in advance by a central server. A mobile host only reports to the server when it steps in or out of one of the monitored regions. This problem is different from the problem investigated in this paper.

Several recent work offer data lookup and delivery techniques for general data in MANET [2], [3], [4], [5], [6]. The data are not necessarily location dependent. A naive approach is for a querying host to broadcast its query to the entire network. Any host having the requested data item replies to the querying host. An alternative is for a host to periodically broadcast an advertisement message about the data it has to the entire network. Hosts store the advertisement message in their cache and use it to answer future queries. Both approaches are very expensive due to network wide broadcasting. Host mobility also outdates cache entries. To reduce the communication overhead, the work in [3] lets each host broadcast its advertisement messages at a different rate. Another technique [5] makes a host send its advertisement

messages along certain geometric trajectories such as West, East, South, and North. Hosts along the trajectories cache and forward the advertisement messages. The querying host sends its query along geometric trajectories that eventually intersect with the advertisement trajectories. The host at the intersection of the advertisement path and the querying path answers the query. This technique, however, still has a scalability problem as indicated by [6] which proposes a solution to address the scalability problem. Their solution works well for low host mobility. However, for moderate to high host mobility, many mobile hosts keep the same advertisement message since for each host movement, new hosts are along the geometric trajectories of the moving host.

In a group-based distributed service discovery protocol [2], each host periodically broadcasts a service advertisement (SA) message in a limited number of hops. Services are grouped into a smaller number of groups. The SA message contains a list of service groups this host's services belong to and a list of non-local service groups this host has seen in its vicinity. Any host receiving the SA message caches the message. The querying host checks its cache for the service it wants. If no information is found in its cache, the querying host broadcasts a service request in a limited number of hops. Hosts receiving the request first uses their cache to determine whether and how to forward the request. This solution reduces the number of messages forwarded among mobile hosts but requires the group information to be included in each advertisement message, which increases the size of the advertisement messages.

Existing data lookup and delivery techniques for general data or service discovery protocols can be applied to solve the problem of sharing location dependent experiences. However, we can expect that these techniques are less efficient since they do not exploit the location information associated with each experience.

In a recent application for sensor networks [11], static sensors report their reading to one moving sink only when the moving sink gets sufficiently close to the sensors. In this application, the mobile sink is required to know its entire travel route and broadcasts its route and speed to the entire network so that the sensors in the network can calculate when they should report their data to the mobile sink. In our case, a mobile host needs not know the entire travel route. Each mobile host only knows its next location and can change its course based on the experiences found in the vicinity of the next intended location. Furthermore, the hosts reporting their experience are mobile, making our problem more complicated than using static sensors to report the sensor readings.

## III. PROPOSED TECHNIQUE FOR SHARING LOCATION DEPENDENT EXPERIENCES

In this section, we describe our technique called *LODE*, an acronym for Location Dependent Experience Sharing. We assume that the boundary of the intended exploration area is known in advance and the mobile hosts only move within the area. Prior to an exploration, each host is configured with the width and height of the exploration area and the virtual grid

size smaller than the exploration area. Hence, a host sees the exploration area as a number of virtual grids (see Fig. 1). Each mobile host is able to determine its own position using some positioning system such as GPS. Each host has a memory and a local disk. The host's memory is small but enough to run our protocol and keep one data item. The local disk is much larger and can store a number of experiences (summaries and data items). All hosts have the same communication bandwidth, transmission range, and memory and disk capacity. LODE uses an underlying broadcast protocol provided by the network layer to broadcast a message within a number of hops limited by the *time-to-live (TTL)* of the message. LODE aims to provide low query response time and low communication overhead.

### A. Preliminaries

We separate an experience into two parts: a *data item* and an *experience summary*. A data item can be text, images, or videos about an interesting event occurring at a location. Data items need not be of the same size. An experience summary has (i) a location where the experience takes place; (ii) time when the experience is recorded; (iii) a unique experience ID; (iv) owner ID; and (v) an optional brief description of the data item. For some application, a mobile host may select only some data items to review based on these descriptions. The experience summary and the corresponding data item of an experience have the same experience ID. The experience ID can be generated from the IP or MAC address of the host and the time the experience is recorded. The owner ID uniquely indicates the host that stores the data item. Mobile hosts can issue two types of queries:

- **Summary query** consists of a) the top-left and bottom-right coordinates of the query rectangle indicating the vicinity of the host's next intended destination and b) the amount of available disk space on the querying host. Note that shapes other than rectangles can also be used for a summary query. We chose a rectangle for ease of explanation. The available disk space is used by the host receiving the summary query to determine whether to send additional information.
- **Data query** specifies the unique experience ID of the data item to be retrieved. The ID is obtained from the summary returned as a result of the summary query.

The core ideas of LODE are as follows. Since a data item can be large, we keep only one copy of a data item in the aggregated storage space of all the mobile hosts. We maintain one or more copies of an experience summary as follows. We let all the mobile hosts that are currently in or about to move into a virtual grid store the summaries of all experiences that have been created in the grid. The last host that would leave a virtual grid empty is required to keep the summaries of this grid with it. Since a summary query specifies a query rectangle, any mobile hosts in the virtual grid enclosing or intersecting the query rectangle reply to the querying host. If the querying host does not receive a reply within a pre-defined timeout period because either no experiences were created in

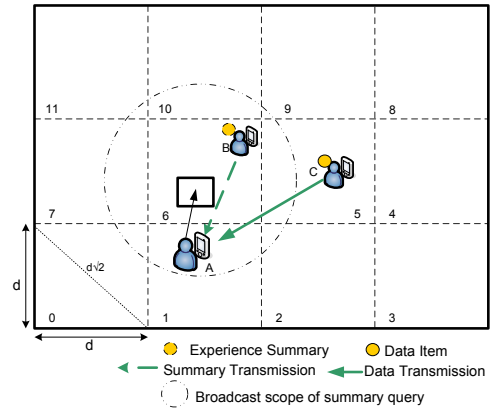


Fig. 1. LODE using Virtual Grids

the query rectangle or the network is partitioned, the host records `EMPTY` as the summary corresponding to the query rectangle.

Fig. 1 illustrates these ideas. Suppose that the user of host A wants to move to a new location. She issues a summary query with the query rectangle indicating the vicinity of the new location (i.e., the rectangle in Grid No. 6 in Fig. 1). Note that the user is unaware of the virtual grids, but the application running on the mobile host of the user is. Host A knows that the query rectangle is not in its current virtual grid. The host broadcasts a summary query for the user's specified query rectangle with a limited TTL and waits for a reply for a pre-defined timeout period. Host B currently in the virtual grid of the query rectangle (Grid No. 6) sends the summaries of all experiences in the query rectangle to host A. Host B has the summaries since we make all hosts in a virtual grid maintains all the summaries of the experiences created in the virtual grid they are currently in.

Next, the user of host A chooses to view one or more data items corresponding to the received summaries. Host A broadcasts a data query in a limited scope to get the desired data item. Note that the location of the owner host of the experience (the one storing the data item) is not part of the summary since hosts are mobile and we do not want to keep updating the summary each time the owner host moves. Host A waits for a reply for a pre-defined timeout period. Host C, the owner host replies to the querying host, host A. If host A is closer to the virtual grid of the query rectangle than host C, host A stores the received data item on its disk and host C deletes the data item from its disk. Host A lets other hosts in the destination virtual grid knows that host A is the new owner of the data item. At most one host replies for a given data query since we ensure that only one host stores the data item in the entire network to effectively utilize the aggregate storage space. The querying host may not receive any reply because of network partitioning. In this case, after the timeout period, the host records `EMPTY_DATA`.

Note that prior to host A issuing the query, host C has moved out of Grid No. 6 after it created the experience in that grid. Before leaving the grid, host A found that at least one host

(host B in the example) was in the grid and can maintain the summaries of all experiences in the grid. Host C then deleted all the summaries of Grid No. 6 from its disk but it still keeps the data item it created in that grid. If host A actually moves into Grid No. 6, this host broadcasts a summary query for the entire Grid No. 6 and stores the returned summaries on its local disk. Subsequent query rectangles in this grid issued by this user can be answered using the on-disk summaries. With summaries and data items within the limited broadcast scope about the size of the virtual grid in most cases, LODE can shorten the response time and reduce communication overhead. Next, we discuss the details of the protocol.

### B. LODE Protocol

At any one time, each host is located in one virtual grid of size  $dxd$  meters<sup>2</sup>. Let  $tr$  be the transmission range (in meters) of a host. Let  $DH = \lceil \frac{d\sqrt{2}}{tr} \rceil$  be the minimum number of hops covering the diagonal distance of a virtual grid. Each mobile host maintains a *FullFlag* variable initialized to false. The host sets this variable to true when it is not able to store a summary broadcast by other hosts in its current grid since its local disk is full. The host performs one of the following actions. Note that we only give the pseudo code to convey the important idea of the technique.

- Create a new experience: After creating an experience in memory, the user tells her mobile host to save an experience on disk.

#### Save\_Experience()

```
If available space is enough for the summary and the data item
    Store both the data item and the summary on disk
    Broadcast the summary with the message TTL set to DH
Else Discard the experience
```

With this TTL setting, all hosts in the broadcasting host's grid receive the summary. Each of these hosts stores a copy of the summary in its local disk if its disk space is available in order to answer summary queries from other hosts that wish to enter this grid. We do not broadcast the data item since it may be large. If the broadcasting host is near the grid border, the summary will reach hosts in a neighboring grid as well, but these hosts will not store the summary outside of their current grid.

- Receive a new experience summary: When a host receives a new summary, it executes the following.

#### Receive\_New\_Summary()

```
If this host is in the same grid as the host originating the broadcast of the summary
    If enough disk space is available to store the new summary
        Store the summary on disk
    Else
        Set FullFlag to true
        Discard all summaries of the host's current grid
    End If
Else Discard the summary
```

Only a host within the broadcast scope of the experience summary and in the same grid as the host originating the

summary broadcast will try to keep the summary. When the host cannot store all summaries of its current grid, it sets *FullFlag* to true and deletes all on-disk summaries of the current grid. This is to utilize its available disk space for storing its own future experiences.

- Enter a neighboring grid: When the user decides to move to her next location, the mobile host checks whether the new location is in a neighboring grid. If so, the mobile host executes the following code that lets the entering host keep summaries of all experiences recorded in the destination grid if the host has enough available disk space.

#### Enter\_Neighbor\_Grid()

```
1)  $GTTL = 2 * DH$ 
2) Repeat
    2.a) Broadcast a summary query with the message TTL set to  $GTTL$ , the query rectangle set to the grid, and the available space set to the current available disk space
    2.b) If not getting a query response within a timeout period
         $GTTL = GTTL + DH$ 
    End If
    Until getting a response or  $GTTL$  exceeding a TTL threshold
3) If not getting a response
    /* network partitioning or no experiences in the destination grid */
    3.a) Set the summary for this grid to EMPTY
4) Else
    If the response is negative
    /* This host's available disk is too small for the summaries */
    4.a) Set FullFlag to true
5) Else
    /* the response is positive */
    5.a) Set FullFlag to false
    5.b) Store the summaries in the response on disk
    5.c) If the responding host (sender) is not in the destination grid of this host
        5.c.1) Receive and store summaries of additional grids on disk.
        5.c.2) Ask the sender to delete from its disk all the summaries it sent to this host
    End If
End If
```

Only one broadcast is needed if there exists a mobile host in the destination grid. Otherwise, more broadcasts (limited by the TTL threshold) are issued to find a host (in a neighboring grid of the destination grid) that has all summaries of experiences of the destination grid. Since several hosts in the destination grid may be responding, the querying host picks the one that responds first and ignores subsequent responses.

When the chosen responding host (sender) is not in the destination grid, the querying host asks the sender to delete the summaries of the destination grid if the querying host has enough space. This is because the querying host will soon be in the destination grid and will answer summary queries related to the destination grid instead. Depending on the amount of the available disk space indicated in the summary query, the sending host may send additional summaries. These summaries

are from experiences created in the grids closer to the destination grid than the sending host's current grid. The sender deletes these summaries from its disk only if the querying host can store these summaries (Step 5.c.1). The idea is to keep the summaries near their original locations.

- Leave the current grid: When a host wishes to leave its current grid, it decides whether to carry summaries related to this grid with it or not. The host will not carry the summaries of its current grid with it if there exists some other host in its current grid. The purpose is to try to keep experience summaries near the locations they were created.

Leave\_Current\_Grid()

- 1) Broadcast a LEAVE message with the message TTL set to  $DH$
  - 2) If this host gets a response within a timeout period
    - /\* Some other host in this grid has the same summaries \*/
    - Delete all summaries related to its current grid
- End If

Any host receiving the LEAVE message executes the following code.

Receive\_LEAVE\_Message()

If the host is in the same grid as the sender and its  $FullFlag$  is false  
Reply to the sender  
End If

Note that hosts in a different grid do not reply and hosts that are in the same grid, but do not have all the summaries of this grid (i.e.,  $FullFlag$  is true) do not reply.

- Process a user query: Before the user moves to a new location, the user sets the desired query rectangle indicating the vicinity of the new location and the mobile host performs the following.

Issue\_Summary\_Query()

- 1) If the query rectangle is entirely in the current grid and  $FullFlag$  is false
    - 1.a) Return only relevant summaries using the on-disk summaries of the current grid
  - 2) Else  
If the query rectangle is entirely in a neighboring grid of the current grid /\* and  $FullFlag$  is true \*/
    - 2.a)  $GTTL = 2 * DH$
    - 2.b) Broadcast a summary query with the message TTL set to  $GTTL$ , the query rectangle set to the user's query rectangle, and the available disk space set to negative
    - 2.c) If receiving a response within a timeout period  
Return the user the summaries in the response  
End If
  - 3) Else
    - 3.a) Find all candidate grids that intersect the query rectangle
    - 3.b) Let  $MAXD$  be the largest distance between the host and the farthest corner of each of the candidate grids
    - 3.c) Broadcast a summary query with the message TTL set to  $\lceil \frac{MAXD}{tr} \rceil$
    - 3.d) If receiving a response within a timeout period  
Return all the summaries in the user's query rectangle  
Else Return EMPTY
- End If

If the query rectangle is entirely within the current grid, the summary query can be answered right away since the host stored all summaries of the grid already on its disk if  $FullFlag$  is false when it entered the grid. Otherwise, the host broadcasts a summary query to grids enclosing or intersecting the user's query rectangle.

Based on the receiving summaries, the user selects an experience to review. The mobile host issues a data query for the user's selected experience as follows.

Request\_Data()

- 1) Set  $GTTL$  to the number of hops covering the diagonal distance between the querying host and the farthest corner of the exploration area.
  - 2) Broadcast a data query for the data item ID with the TTL set to  $GTTL$
  - 3) If receiving the data item within a timeout period
    - 3.1) If the host has enough disk space to store the requested data item and the sender's grid is not in the same grid as the query rectangle
      - /\* move the data item closer to its original location \*/
      - 3.1.a) Store the data item on its disk.
      - 3.1.b) Ask the sender to delete the data item from its local disk.
      - 3.1.c) Broadcast the updated summary with the message TTL set to  $DH$  to announce that this host is the new owner of the data item.
- End If  
Else Return EMPTY\_DATA

The data owner sends the data item to the querying host using some existing routing protocol such as AODV. If the querying host has enough space to store the data item, the querying host asks the current data owner to delete the item from its local disk. The querying host becomes the new owner of this data item. The idea is to maintain a data item in a mobile host closer to the location that the data item was created. A querying host issues one data query to request one data item. We can reduce the overhead further by packing several data item IDs in one data query.

- Receive a summary query: The host receiving the summary query executes the following code. We use the term destination grid to refer to the grid of the query rectangle.

Receive\_Summary\_Query()

- 1) If the host does not have summaries related to the query rectangle or  $FullFlag$  is true  
Discard the summary query  
Else
- 2) If this host is in the same grid as the query rectangle  
Send the querying host all summaries of experiences in the query rectangle
- 3) Else  
If this host and the query rectangle are in two different grids
  - 3.a) Mark a response as negative if the querying host's available space is smaller than the total size of the experiences in the destination grid
  - 3.b) Put all summaries of experiences in the destination grid in the response if the response is not marked negative
  - 3.c) Send the response to the querying host
  - 3.d) When asked by the querying host to delete the summaries of a particular grid, this host deletes all on-disk summaries related to the specified grid

- 3.e) Send all summaries related to each of the neighboring grids of the destination grid if the distance between the neighboring grid of the destination grid is closer to the destination grid than this host's current grid and the querying host has enough available disk space to store all these summaries

End If

The rationale behind Step 3) is to keep the summaries of experiences near the location where the experiences were created.

- **Receive a data query:** When receiving a data query, the host executes the following code.

Receive\_Data\_Query

- 1) If the host has the requested data item
    - Send the requested data item
    - If this host is not in the same grid as the querying host
      - Delete this data item when asked by the querying host
- End If

The host with the data item deletes its data item only when the querying host can store the data item. (See Step 3.1.b in *Request\_Data()*).

#### IV. PERFORMANCE STUDY

In this section, we investigate the performance of LODE compared with those of 1 stationary server (1-SS) and 4 stationary servers (4-SS). We implemented simulators for the three techniques. For LODE, each host has limited disk space that can store ten experiences on average. For 1-SS and 4-SS, each stationary server has very large disk space able to keep all experiences reported by all mobile hosts. The 1-SS scheme uses one stationary server placed at the center of the exploration area. The 4-SS scheme uses four stationary servers distributed uniformly in the exploration area. All four stationary servers communicate with each other through their own network to exchange experiences reported from all mobile hosts. In our study, we ignore the communication cost among these stationary servers themselves. In 1-SS and 4-SS, each mobile host submits its queries or reports its new experience to the nearest stationary server at the time via multi-hop relays.

We compare the three techniques using the following performance metrics.

- **Summary query latency** measured as smallest number of hops needed to reach the first host having the answer for a summary query. Recall that the summary query indicates the area (query rectangle) where the user wants to know any prior experience.
- **Data query latency** measured as smallest number of hops needed to reach the first host having the requested data item for a data query.
- **Experience success rate** measured as a fraction of the total number of users' queries that are satisfied. A user's query is satisfied if the user can successfully retrieve the data items of all experiences in a query rectangle. The user may not be able to get some experience either

because the summary is not found or the data item is not found due to network partitioning.

- **Drop rate** measured as a fraction of times a new experience cannot be stored in a mobile host's disk due to limited disk space at the host. For the 1-SS and 4-SS schemes, the drop rate is always zero since stationary servers have enough disk space to keep all the experiences.

Table I shows the default parameter values for our simulations. We generate a set of trace files for various simulation parameter values. For each trace, we randomly place a number of mobile hosts and a number of events in an exploration area. We randomly generate the size of the data item part of an experience between 1 and 10 MB. Each host first empties its disk and determines its next location. The host movement pattern follows the M-GRID model [12] that simulates a metropolitan environment with roads and junctions made up of square grids. Using this movement pattern, a host either stops for a period of time or moves in one of the four directions along a grid line at a randomly chosen speed. Before moving to a new location, a host indicates a query rectangle with the new location at the center of the query rectangle to get all summaries as well as data items of experiences observed in the query rectangle. A host records a new experience when it passes by an event (within 1 *meter* from the event location) for the first time. Each trace file records the host locations and summary queries issued by these hosts. All three simulators used the same set of trace files. After reading a trace file to get movement and query patterns, the simulator records the performance metrics after some warm up time period (100 simulation steps). Simple flooding is used for broadcasting. Our simulators implement a Null MAC layer (i.e., a host gets a clear channel when it wants to broadcast). Each point in the plots in the next subsections is the average over thirty simulation runs.

TABLE I  
SIMULATION PARAMETERS

Parameter	Default value	Unit
Exploration area	160 × 160	<i>meter</i> <sup>2</sup>
Virtual grid area	10 × 10	<i>meter</i> <sup>2</sup>
M-GRID	2 × 2	<i>meter</i> <sup>2</sup>
Host density	5	<i>neighbors/host</i>
Event density	0.5	<i>event/meter</i> <sup>2</sup>
Transmission radius	10√2	<i>meter</i>
Query rectangle	1 × 1	<i>meter</i>
Host's moving speed	0-10	<i>meter/s</i>
Average size of a data item	5	<i>MB</i>

##### A. Effect of Host Density

We investigate the effect of various host densities on the success rate for retrieving experiences and average latencies to get responses for summary queries and data queries. We varied the host density between 1 and 10 average neighbors per host and fixed the other parameters at their default value shown in Table I. The results are plotted in Fig. 2. Increasing

the host density improves the network connectivity. When the host density is at least 5 neighbors per host on average, the network is connected in our study. With a connected network, a querying host can always retrieve an existing experience created by another host.

Fig. 2(a) shows that the success rate of retrieving experiences (both summaries and corresponding data items) increases as the host density increases. When the host density is at least 5 neighbors per host, all schemes offer the highest experience success rate. For a lower host density, the success rate of LODE is higher than that of 1-SS but lower than that of 4-SS. The 1-SS scheme requires each host to communicate with one server that is about 10 hops away on average (see Fig. 2(b)). A disconnection in any one of these hops results in a failure. Since hosts in 4-SS are closer to one of the stationary servers, there is a less chance for a disconnection compared to 1-SS. In LODE, querying hosts are closer to hosts having the answers compared to 4-SS (see Fig. 2(b-c)). However, LODE has a slightly lower success rate than 4-SS because the success rate of LODE depends on the success of finding both the summaries and the corresponding data items. Next, we investigate the average latency for a summary query and a data query, taking into account only the cases the querying host gets a response in return.

Fig. 2(b) demonstrates that LODE offers the smallest latency for summary queries, which is significantly lower than those of the 1-SS and 4-SS schemes. The latency reflects the average distance between mobile hosts and hosts having the summary answers for the queries. The 1-SS scheme offers the worst latency since only the server at the center of the exploration area has the answers for the queries. The 4-SS scheme offers a lower latency than the 1-SS scheme does because hosts are closer to one of the four stationary servers. LODE incurs the smallest latency because of the following reasons. If the query rectangle is within the querying host's current virtual grid and *FullFlag* is false, the host needs not broadcast the summary query since in many cases the summaries of all the experiences in the current grid have already been downloaded in this host's disk when it has first entered the grid. The querying host broadcasts a summary query only when the query rectangle is not entirely in its current virtual grid or the *FullFlag* is true. However, if the query rectangle involves neighboring virtual grids. Hosts in the neighboring grids can answer the query quickly since LODE lets hosts in a virtual grid keep the summaries of all experiences observed in the virtual grid. Using more stationary servers (8 or 16 servers) could reduce the latency lower than that offered by LODE, but requires more servers.

In terms of the average latency for a data query, Fig. 2(c) shows that LODE offers the lowest latency compared with the schemes using stationary servers since it tries to maintain the data items near their originating locations. As seen in Fig. 2(b-c), the latencies offered by the three schemes are insensitive to the host density. This is expected as the average of the smallest distance between mobile hosts and stationary servers in the 1-SS and 4-SS schemes or between querying hosts and

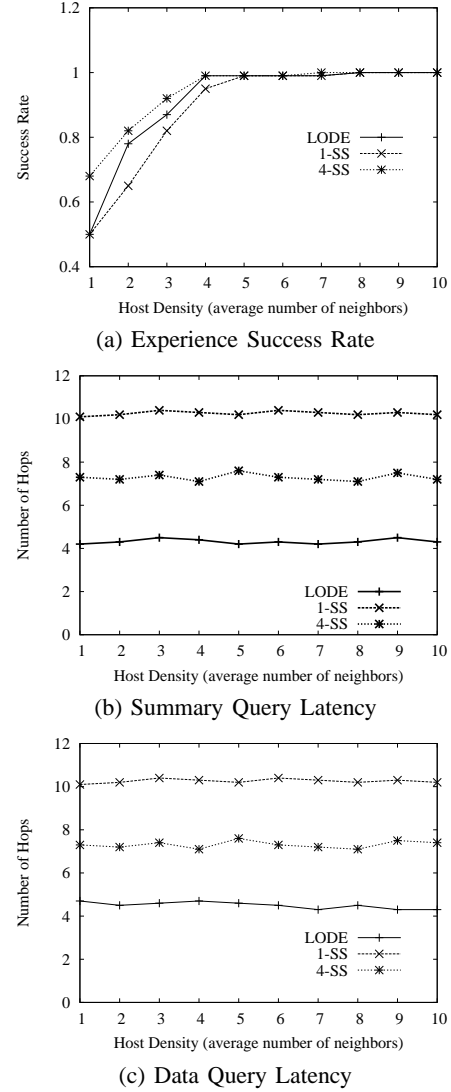


Fig. 2. Effect of Host Density

other hosts having the answers does not change according to the host density.

### B. Effect of Event Density

We generated a number of events, ranging from 0.1 to 1 *event/meter*<sup>2</sup> and set other parameters at the default values. Fig. 3(a) shows the effect of various event densities on the experience drop rate and the latencies for summary queries and data queries. As the event density increases, more mobile hosts are not able to store new experiences from these events in LODE due to hosts' limited disk space. Hence, the experience drop rate increases. When the event density is 0.5, less than 10% of the experiences are discarded. The experience drop rate is zero for the 1-SS or 4-SS schemes since each stationary server has large disk space that can keep all the experiences. Fig. 3(b-c) shows that LODE offers the least latency among all the techniques. This is because LODE tries to keep experiences in mobile hosts near the originating

location of each experience. LODE does not increase latencies even when more experiences (as a result of more events) are observed, which makes the scheme desirable.

### C. Overhead

The communication overhead of the three schemes is dominated by the overhead of the underlying broadcast protocol. In 1-SS and 4-SS schemes, the average distance between the querying host and the hosts having the answers is at least 1.5 times that of LODE (Fig. 2 and Fig. 3), which means more communication overhead when using simple flooding. To reduce the flooding overhead, for moderate to high host mobility, some GeoCast protocol [13] can be employed since the locations of the stationary servers are known. Similarly, a GeoCast protocol can be used with LODE when a host issues a summary query to the virtual grid of the query rectangle. The performance gap in terms of numbers of forwarded packets by all hosts heavily depends on the underlying broadcast protocol used for each technique. That is, the performance gap when using simple flooding is wider than that when using another efficient broadcasting technique.

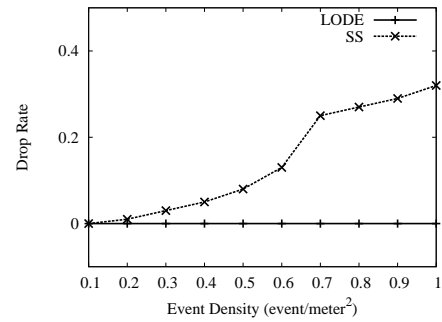
The virtual grid size also impacts the performance and overhead of LODE. A too large virtual grid will result in more communication overhead and more disk space needed to keep all the summaries in the virtual grid. A too small grid size results in more probabilities of relocations of summaries and data items. Due to limited space, we omit the study of the tradeoffs between the grid size and the overhead.

## V. CONCLUDING REMARKS

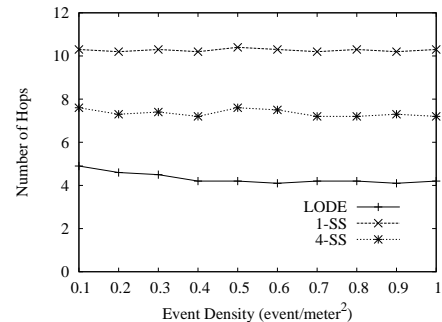
We have presented a new problem of sharing location dependent experiences for applications such as hurricane response missions, combat zones, or deep sea or deep space exploration. We propose a new distributed solution called LODE and show that it cuts the latency for accessing desired experiences by more than half compared with the technique in which all mobile hosts report to a single stationary server for any new experiences they encounter. Our future works include the detailed study of the impact of the virtual grid size, a further reduction of broadcast overhead for data items by predicting the location of a host after some time steps via analysis given a movement pattern. This will enable us to determine the appropriate TTL value for a data query. We will also evaluate the performance of LODE for other hosts' movement patterns such as the random way points model.

## REFERENCES

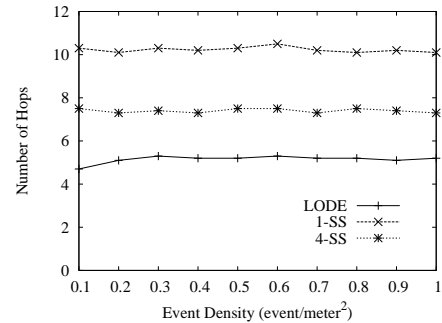
- [1] H. Zhou, "A survey on routing protocols in MANETs," *Technical report: Michigan State University: MSU-CSE-03-08*, 2003.
- [2] D. Chakraborty, A. Joshi, and T. Finin, "GSD: a novel group based service discovery protocol for MANETs," in *Proc. of IEEE MWCN*, 2002.
- [3] S. H. Shah, K. Chen, and K. Nahrstedt, "Cross layer design for data accessibility in mobile ad hoc networks," in *Proc. of SCI 2001*, Orlando, Florida, 2001.
- [4] E. Guttman, "Service Location Protocol: Automatic discovery of IP network services," *IEEE Internet Computing*, vol. 3, no. 4, 1999.
- [5] I. Aydin and C.-C. Shen, "Facilitating match-making service in ad hoc and sensor networks using pseudo quorum," in *Proc. of IEEE ICCCN*, 2002.



(a) Experience Drop Rate



(b) Summary Query Latency



(c) Data Query Latency

Fig. 3. Effect of Event Density

- [6] J. B. Tchakarov and N. H. Vaidya, "Efficient content location in wireless ad hoc networks," in *Proc. of IEEE MDM*, 2004, pp. 74–85.
- [7] H. M. O. Mokhtar and J. Su, "Universal trajectory queries for moving object databases," in *Proc. of IEEE MDM*, 2004, pp. 133–144.
- [8] D. Pfoser, C. Jensen, and Y. Theodoridis, "Novel approaches to the indexing of moving object," in *Proc. of VLDB*, Cairo, Egypt, 2000.
- [9] Y. Cai, K. A. Hua, and G. Cao, "Processing range-monitoring queries on heterogeneous mobile objects," in *Proc. of IEEE MDM*, 2004.
- [10] S. Prabhakar, Y. Xia, D. Kalashnikov, W. G. Aref, and S. Hambrusch, "Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects," *IEEE Transactions on Computers*, vol. 51, no. 10, pp. 1124–1140, October 2002.
- [11] G. X. C. Lu, O. Chipara, C. Fok, and S. Bhattacharya, "A spatiotemporal query service for mobile users in sensor networks," in *Proc. of IEEE ICDCS*, 2005.
- [12] B.-S. Lee, K. J. Wong, B.-C. Seet, L. Zhu, and G. Liu, "Performance of mobile ad hoc network in constrained mobility pattern," in *Proc. of ICWN 2003*, Las Vegas, Nevada, 2003.
- [13] P. Yao, E. Krohne, and T. Camp, "Performance comparison of Geocast routing protocols for a MANET," in *Proc. of IC3N*, 2004, pp. 213–220.