HERO: Online Real-time Vehicle Tracking in Shanghai*

Hongzi Zhu[‡], Yanmin Zhu[§], Minglu Li[‡], and Lionel M. Ni[§]

[‡]Shanghai Jiao Tong University
Shanghai, China
{hongzi, mlli}@sjtu.edu.cn

[§]Hong Kong University of Science and Technology
Hong Kong, China
{zhuym, ni}@cse.ust.hk

***Abstract*—Intelligent transportation systems have become increasingly important for the public transportation in Shanghai. In response, ShanghaiGrid (SG) aims to provide abundant intelligent transportation services to improve the traffic condition. A challenging service in SG is to accurately locate the positions of moving vehicles in real time. In this paper we present an innovative scheme HERO to tackle this problem. In SG, the location information of individual vehicles is actively logged in local nodes which are distributed throughout the city. For each vehicle, HERO dynamically maintains an advantageous hierarchy on the overlay network of local nodes to conservatively update the location information only in nearby nodes. By bounding the maximum number of hops the query is routed, HERO guarantees to meet the real-time constraint associated with each vehicle. Extensive simulations based on the real road network and trace data of vehicle movements from Shanghai demonstrate the efficacy of HERO.**

Keywords-vehicle tracking; spatiotemporal locality; real-time system; RFID system; peer-to-peer network

I. INTRODUCTION

Intelligent transportation systems (ITSs) have been evolving rapidly in the past two decades, leveraging advanced computing and communication technologies. ITSs help coordinate traffic condition, improve safety, reduce environmental impact, and make efficient use of available resources. Shanghai, the largest metropolis in China, covers an area of 5,800 square kilometers and has a large population of 18.7 million. The economy of Shanghai is soaring today and the growing traffic has become a serious challenge. In response to the challenge and the needs of the public, the Shanghai government has established the ShanghaiGrid (SG) project since 2005, with the ambitious goal of building a metropolitan-scale traffic information system. This project will construct the basic infrastructure, composed of a great number of traffic information collectors and local information processing nodes. The location and status information of vehicles can be actively captured by these pervasively deployed collectors and further logged and processed by local nodes interconnected through the Internet. The goals of the project are two-fold. First, it tries to make the available transportation infrastructure to be used more efficiently. Second, it aims to provide the public with a wide spectrum of ITS applications [8], ranging from navigation, trip planning and optimal route selection to congestion avoidance and bus arrival prediction.

Among all the others, *online real-time vehicle tracking* is a fundamental service in SG, which refers to tracking the current position of a certain vehicle in real time. A wide spectrum of compelling applications can be implemented on top of this basic service. For example, users will be authorized to track individual vehicles that they are concerned with, such as their own or friends' cars, public buses and taxies. In particular, there exist several critical types of vehicles in the city, which need to be located urgently, such as stolen cars, speeding cars, ambulances and police cars. Besides these application scenarios, it is also an indispensable building block underpinning many other high-level applications. For example, in the *bus arrival prediction* application, the tracking service can be used to locate the nearest feasible bus.

However, real-time vehicle tracking in the metropolitan-scale system is very challenging because of several rigorous requirements. First, users (or high-level applications) often pose a real-time requirement on tracking a certain vehicle. That is, any query for the vehicle must be answered within a certain bounded time. Otherwise, the returned answer may become invalid or useless. For example, a query tries to locate the current location of a stolen car. If the query fails to be answered within a short time, the car could actually be far away from the returned location because it may be moving at a high speed. Second, the system should be scalable to support up to millions of users and hundreds of thousands of vehicles. The huge number of simultaneous queries is a serious issue. Third, the system should be robust to node failures. In such a large-scale distributed system consisting of thousands of local nodes, system maintenance is not a trivial issue.

To realize this service, a centralized scheme is straightforward, where location information of all vehicles is sent back to a centralized database and constantly maintained. A user, who wants to track a vehicle, can send a query to the central server. The server then processes the query and returns the vehicle's information to this user. However, it is infeasible for the metropolitan-scale system due to the huge amount of vehicle data streams. For example, there are 22,413 crossroads in Shanghai. Even in 2001, the average number of vehicles running across a crossroad per minute in daytime was up to 33

* This research is supported by the National Basic Research Program (973 Program) of China (No. 2006CB303000), NSFC (No. 60473092, 90612018 and 60533110), STCSM (No. 05DZ15005), Hong Kong Research Grant Council (HKUST6183/05E) and Intel Research Council.

[1]. This produces in total about 12 thousand events per second. Such a large volume of location updating data can easily overwhelm the centralized server.

Therefore, it necessitates efficient designs of distributed solutions. As an alternative scheme, captured vehicle information can be stored locally at distributed nodes. Therefore, there is little updating data as required in the centralized scheme. Nevertheless, by this means, there is no hint about the enquired vehicle for a query. To track the vehicle, an intuitive scheme is to flood the query across the network which can always locate the desired vehicle. However, flooding search not only incurs a large amount of network traffic and hence is subject to poor scalability, but also fail to satisfy the rigid real-time requirement. To reduce query traffic, there is another search scheme based on random walks, which introduces modest network traffic. However, this scheme is limited by the problem of unbounded response latency of the query. As a result, there is no existing successful solution, to the best of our knowledge, to tracking vehicles in real time in a large-scale distributed system.

In this paper, we propose a novel scheme Hierarchical Exponential Region Organization (HERO) which satisfies the unique requirements of real-time vehicle tracking in a metropolitan-scale distributed system. In essence, HERO connects local nodes into an overlay network matching the underlying road network. A further hierarchical structure over the overlay network is then constructed and dynamically maintained while the vehicle is moving along. Exploiting the inherent spatiotemporal locality of vehicle movements in the urban setting of Shanghai, this hierarchy enables the system to conservatively update location information of a moving vehicle only in nearby nodes. The distinctive features of HERO are twofold. First, it guarantees that any query, which can be injected anywhere in the city, can meet the real-time constraint associated with each vehicle, by bounding the maximum number of hops that the query is routed. Second, it significantly reduces the communication overhead of both location updating and query routing, and therefore is truly scalable to support hundreds of thousands of vehicles and millions of users. Moreover, HERO is a fully distributed light-weight protocol extensible to the increasing scale of the system. In addition, it is robust to node failures and also able to tolerate inaccurate location readings.

The original contributions that we have made in the paper are highlighted as follows:

- We propose a novel protocol HERO for real-time vehicle tracking in a metropolitan-scale intelligent transportation system. HERO employs a distributed technique to store the large volume of vehicle information. It guarantees to meet the real-time constraint associated with a vehicle for answering any query about the vehicle, and is truly scalable with respect to the number of vehicles, the number of queries and the system scale.
- We conduct in-depth theoretical analysis, identify the tradeoff relationship between the communication overhead and the query response time, and draw an optimal configuration of system parameters of HERO, minimizing network traffic under real-time constraints.
- We evaluate the performance of the HERO approach through precise trace-driven simulations. We base our simulations on the real road network and trace data of vehicle movements in Shanghai, and compare the performance of HERO with other alternative schemes.

The rest of this paper is structured as follows. Section II compares HERO with related work. In Section III, we introduce the infrastructure that will be deployed in the SG project. Section IV elaborates the design of HERO and presents theoretical analysis for the optimal configuration of the protocol parameters. Several design issues that may be encountered in practice are discussed in Section V. In Section VI, we introduce the trace-driven methodology that we use to evaluate the performance of HERO and present simulation results. Finally, we present concluding remarks and outline the directions for future work in Section VII.

II. Related Work

The Globe system [9] constructs a static world-wide search tree for mapping object identifiers to the locations of moving objects. It is not flexible to expand or adjust the structure and may have the bottleneck problem near the root of the directory tree structure. Alminas et al [10] introduce a distributed approach for load balance but haven't taken the number of queries into consideration.

In database community, indexing techniques have been proposed for tracking moving objects but they are based on the assumption of the existence of centralized databases [11-14]. Despite the large number of existing methods, there is no applicable one for update-intensive applications, where it is infeasible to continuously update the index and process queries at the same time [15]. HERO does not need any centralized database and all routing information is distributed to every node in the system.

In structured peer-to-peer (P2P) networks, various DHT schemes have been proposed to map objects to peers in a decentralized way, thus enabling the system to satisfy queries efficiently [16-19]. However, DHTs may cause large computation and traffic overhead for a large number of rapid updates of moving objects. In unstructured P2P networks, the most typical query methods are based on flooding [7]. Using flooding is not scalable. Several randomized approaches, such as random walks [20, 21] and randomized gossip-based methods [22, 23], have been introduced to distribute and locate objects. Random schemes incur notable long query latency before finding the results in a stable network. HERO introduces minimal updating cost to guarantee the real-time constraints desired by the applications.

III. System Description

As RFID technology continuously evolves, it has been widely used in tracking various mobile objects, such as vehicles [2]. The US government also enacts the TREAD Act

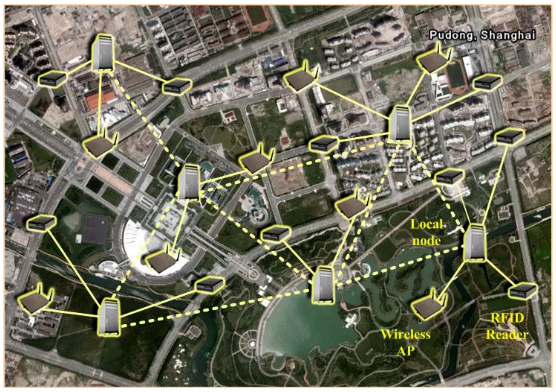

Figure 1. The infrastructure of ShanghaiGrid; a small part of the Pudong District of Shanghai is shown.

[4] which demands RFID tags to be planted in every new tire before September 2007. The SG project exploits the promising RFID and local-area wireless communication technologies. The infrastructure of SG, which is still underway, is illustrated in Fig. 1. RFID readers and wireless APs will be deployed throughout the urban area of Shanghai, typically installed at crossroads. A local node is responsible for collecting data from several close RFID readers and wireless APs within its own domain, and accepts queries from nearby users or applications. A local node is basically a server which connects to the Internet through a dedicated connection.

In the initial prototype of SG, a vehicle is captured using active RFID technology. An active RFID tag emits its ID at a fixed interval and has an effective communication range of about 2 to 80 meters. The battery can sustain the operation of an active RFID tag for about 6 years [3]. A moving vehicle attached with an active RFID tag can be captured if the emitted signal reaches some reader. In addition, a vehicle can actively push important vehicle status information, such as vacancy status, to local nodes by communicating with wireless APs.

As another initial pilot effort in Shanghai, certain vehicles (around 4,000 taxies and 2,000 buses) are equipped with Global Positioning System (GPS) receivers, which can provide coarse-grained location information. A vehicle actively reports its location information back to a centralized database through a wireless cell-phone data channel (i.e., GPRS). Several crucial reasons prohibit this initial effort from being extended for vehicle tracking in SG. First, with crowded high buildings squeezed along most of the narrow streets in the city, it is very difficult for the GPS system to work accurately without any other assistant devices. It is often the case that the reported GPS position of a vehicle can be more than 100 meters deviated from its actual location. To make things worse, a large number of major roads are covered by viaducts which prevent satellites from seeing the vehicles running under viaducts. Second, the intervals of location information reports can be notably long. Due to the GPRS communication cost for transmitting the GPS location information back to the data centre, drivers prefer to choose relatively large intervals. The typical value would be from 1 minute up to 3 minutes. Third, the expense of a GPS receiver as well as data communication cost is quite high. However, the trace data of vehicle movements in the urban area of Shanghai obtained from this prototype using GPS technology is very valuable for study of traffic conditions. We evaluate HERO using the real trace data.

IV. DESIGN OF HERO

In this section, we first give an overview of the HERO protocol, introducing its basic rational. Next, we describe the conservative location updating based on the assistance of a dynamically maintained hierarchy. Finally, we discuss the optimal configuration of the protocol parameters of HERO.

A. Design Overview

There are two critical issues in real-time vehicle tracking. First, the system should limit the maximum query response time to guarantee the posed real-time constrains. Second, the system should minimize network traffic to support a large number of vehicles and queries.

However, there is an intrinsic tradeoff between network traffic and query response time in vehicle tracking. By aggressively updating location information of a vehicle to all the other nodes, the system provides minimal query response time. In contrast, the system suffers from long query response time if the system does not perform any location updating at all. In general, more rigid real-time requirement on tracking a vehicle implies higher network traffic overhead.

Recognizing the inherent spatiotemporal locality of vehicle movements, HERO elegantly manages to solve the two critical issues in an integrated approach. The core idea of HERO is to dynamically update location information of a moving vehicle to all the nodes in the system *in a controlled way*. Generally, the nodes closer to the vehicle are updated more frequently than those further from it and, therefore, have more accurate information about the current location of the vehicle. By this means, HERO significantly reduces location updating cost. Upon receiving a query, the node unlikely has the exact information. However, it knows some other node which has more accurate information about the vehicle and thereby closer to the vehicle. Thus, it forwards the query to that node. Following an elaborately organized routing path, the query can eventually reach the destination node, which keeps the most updated information of the vehicle. The typical latency between two nodes can be easily measured. Thus, by bounding the maximum number of hops that the query is routed, HERO can also meet the real-time constraint for the vehicle.

The key to the design of HERO is how to realize the controlled location updating while bounding the maximum number of hops a query is routed. To achieve this, HERO integrates four effective components:

Overlay construction: To exploit the locality of vehicles' movements, HERO organizes local nodes into an overlay network that matches the underlying real road network in Shanghai (as depicted in Fig. 1, dashed lines present the overlay connections of local nodes). There is a connection between two geographically neighboring local nodes if there is a road between the two corresponding regions. This overlay is easy to build and maintain although the system scale is very

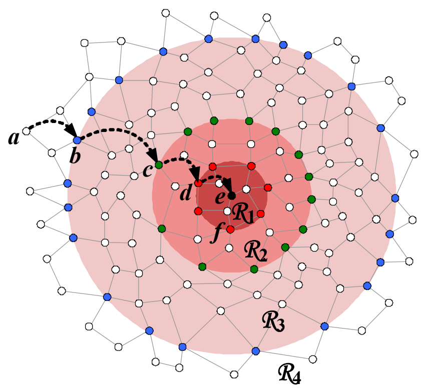

Figure 2. Illustration of hierarchical regions and query routing.

large, with each node having to know its neighbors. To enhance the reliability of the overlay network, additional overlay connection may also be added for two nodes that are geographically close to each other although they are not connected by a real road.

Hierarchy initialization: For every vehicle, HERO divides local nodes into different *regions* which constitute the hierarchy over the overlay network. The regions are organized in the following way, as illustrated in Fig. 2. The first region ($\mathcal{R}_1$) has the smallest size and covers the vehicle. For the example in Fig. 2, $\mathcal{R}_1$ covers node *e*, which is closest to the vehicle and has the latest information about it. The second region ($\mathcal{R}_2$) has a larger size and covers $\mathcal{R}_1$. More generally, a region ($\mathcal{R}_i$) has a larger size than the immediate inner region ($\mathcal{R}_{i-1}$) and covers it.

Restricted location updating: When the vehicle is moving within $\mathcal{R}_1$, the location updating only needs to be carried out among the small set of nodes in $\mathcal{R}_1$. When the vehicle is moving out of $\mathcal{R}_1$, the location updating is extended to more regions. In this case, part of the hierarchy needs to be re-organized. This reorganization aims to restrict location updating within $\mathcal{R}_1$ in the future as much as possible, thereby minimizing network traffic cost for location updating.

Query Routing: With the hierarchy and restricted location updating, an inner region always has more up-to-date location information of the vehicle than outer regions. In HERO, each node has a pointer pointing to a boundary node of its immediate inner region. A query can be issued from any node in the system. For the example in Fig. 2, node *a* receives a query. Node *a* will forward the query to *b*. The query will further be forwarded by node *c* and *d*, and eventually arrives at *e*. Node *e* will return the location information directly back to node *a*. To restrict the maximum number of hops that the query is routed, we limit the number of regions that the hierarchy for the vehicle contains.

In the following subsections, we first describe the process of hierarchy initialization when a new vehicle is joining the system. Next, we describe the detailed mechanism for restricted location updating while the vehicle is moving based on the established hierarchy. Finally, the optimal configuration of design parameters is discussed.

B. *Hierarchy Initialization*

The first node that captures a new vehicle triggers an initialization procedure to establish the hierarchy for the vehicle. As the vehicle may move towards any direction, a region is designed as a disk on the overlay network. Note that, the deployment of local nodes is not necessary to be uniform in the city. They can be more densely deployed where more refined tracking accuracy is required. We will discuss more on this in Section V. In the rest part of this paper, without explicit specification, distance is measured in terms of hops in the overlay network. Each region $\mathcal{R}_i$ has a radius r_i (in hops). A node, which has a distance *d* from the first node, belongs to region $\mathcal{R}_k$ if this region is the smallest one that covers the node. The radius r_k of $\mathcal{R}_k$ is,

$$r_k = \min_{i=1}^{h}\{r_i,\ r_i \geq d\}, \tag{1}$$

where *h* is the maximum number of regions in the system. If *d* equals to certain r_i, $1 \leq i \leq h$, the node is on the boundary of r_i. For query routing, every node maintains a pointer that points to a node which is on the boundary of the immediate inner region. We call this pointer *next-insider*.

To establish the region hierarchy and *next-insider* pointers, the first node initiates an *initialization packet* which contains a `router` field for setting up *next-insider* pointers and a `journey` field for maintaining the distance that the packet has traversed. The first node initializes `router` and `journey` to its own IP address and one, respectively. Then the first node floods the initialization packet throughout the network. Upon receiving the packet, a node first sets its *next-insider* to `router` contained in the packet. Then it checks `journey` in the packet. If `journey` equals to the radius of certain region r_i, the node modifies `router` in the packet to its own IP. It then marks itself as a boundary node of region $\mathcal{R}_i$. Otherwise, it leaves the `router` field unchanged. Next, it increases `journey` in the packet by one and re-broadcasts the packet to its neighbors. In addition, duplicated packets with larger `journey` are silently dropped. After the initialization procedure terminates, the regions are centered at the first node and the hierarchy is established with each node having its *next-insider* set up (as illustrated in Fig. 2).

C. *Restricted Location Updating*

When the vehicle is moving in the city, its information is captured by the local nodes it passes by. When a node captures the vehicle (we call this node *chaser*), it performs location updating, and maintains the hierarchy for the vehicle if necessary. There are three cases depending on the position where the chaser is located in the hierarchy. For presentation clarity, we define a node as a *boundary node* of $\mathcal{R}_i$ if it is a most outer node within $\mathcal{R}_i$. The nodes in $\mathcal{R}_i$ except boundary nodes are *interior nodes* of $\mathcal{R}_i$.

Case 1: the chaser is an interior node within $\mathcal{R}_1$. In this case, the hierarchy for the vehicle remains unchanged. The

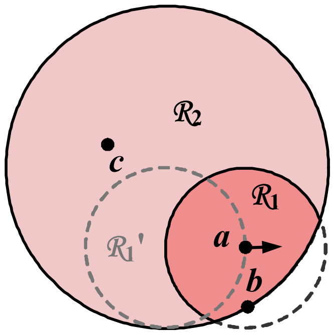

Figure 3. Reconstruction of $\mathcal{R}_1$, node *a* is the chaser and is a boundary node of the first region ($\mathcal{R}_1$').

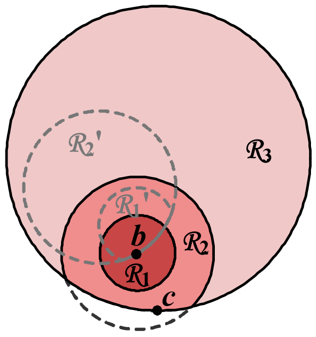

Figure 4. Reconstruction of $\mathcal{R}_1$ and $\mathcal{R}_2$, node *b* is the chaser and also is a common boundary node of the first and second region ($\mathcal{R}_1$' and $\mathcal{R}_2$').

chaser floods the location information of the vehicle to all the other nodes in $\mathcal{R}_1$.

Case 2: the chaser is a boundary node of $\mathcal{R}_1$. In this case, it is possible that the vehicle will move out of $\mathcal{R}_1$ shortly. For example in Fig. 3, node *a* is the current chaser which is a boundary node of $\mathcal{R}_1$ (the dashed circle). When the vehicle moves along the depicted direction, $\mathcal{R}_1$ will not cover the vehicle any more. Two consequences follow. First, a future query cannot be routed to the chaser properly because the information on the boundary nodes of $\mathcal{R}_1$ is out-of-date. Second, to enable the proper routing of a future query, the chaser has to flood the location information of the vehicle to $\mathcal{R}_2$ every time, which will incur larger network traffic overhead. Therefore, HERO needs to re-organize $\mathcal{R}_1$.

To this end, the chaser initiates an *update packet* in which its `router` and `journey` is initialized to its own IP address and one, respectively, as in an initialization packet. The update packet includes an additional `scale` filed that is used to indicate the area that the update packet should be propagated to. In this case, the chaser floods the packet within $\mathcal{R}_2$ by letting the boundary nodes of $\mathcal{R}_2$ stop the flooding. On the one hand, the new $\mathcal{R}_1$ is rebuilt from the current *chaser* within $\mathcal{R}_2$. At the same time, location information is also updated in the new $\mathcal{R}_1$. On the other hand, it updates nodes in $\mathcal{R}_2$ about the current position of the new $\mathcal{R}_1$.

There is a special situation during the reconstruction of $\mathcal{R}_1$, where the new $\mathcal{R}_1$ is truncated by the boundary of $\mathcal{R}_2$. This happens when the chaser is near the boundary of $\mathcal{R}_2$ (e.g., node *a* in Fig. 3). In this situation, a boundary node of $\mathcal{R}_2$ (e.g., node *b*) sets itself as a boundary node of both $\mathcal{R}_1$ and $\mathcal{R}_2$. We call such a node a common boundary node of $\mathcal{R}_1$ and $\mathcal{R}_2$. In this case, $\mathcal{R}_1$ is no longer a disk because it is restricted in $\mathcal{R}_2$. But, this does not affect the operation of our protocol.

Case 3: the chaser is a common boundary node of several regions $\mathcal{R}_1$, $\mathcal{R}_2$, … , $\mathcal{R}_j$ (j>1). This is actually a more general situation of Case 2. This situation results from constant reconstructions of regions as the vehicle is moving. In this case, it is possible for the vehicle to move out of all the regions from $\mathcal{R}_1$ to $\mathcal{R}_j$. The system needs to re-organize regions from $\mathcal{R}_1$ to $\mathcal{R}_j$. For example in Fig. 4, the situation occurs if node *b* is the current chaser, where *b* is also a common boundary node of $\mathcal{R}_1$ and $\mathcal{R}_2$ (the dashed circles).

To re-build regions from $\mathcal{R}_1$ to $\mathcal{R}_j$, the chaser floods an update packet within $\mathcal{R}_{j+1}$. As a result, all regions from $\mathcal{R}_1$ to $\mathcal{R}_j$ are re-constructed within $\mathcal{R}_{j+1}$. In addition, the location information of the vehicle within $\mathcal{R}_{j+1}$ is updated. Similar to Case 2, there is also a special situation during the reconstruction of regions from $\mathcal{R}_1$ to $\mathcal{R}_j$, where several regions, say from $\mathcal{R}_i$ to $\mathcal{R}_j$ might be truncated by some boundary nodes of $\mathcal{R}_{j+1}$. Such a boundary node of $\mathcal{R}_{j+1}$ sets itself as the common boundary node of regions $\mathcal{R}_i$, $\mathcal{R}_{i+1}$,…, $\mathcal{R}_{j+1}$, ($1 \le i \le j$). For example in Fig. 4, node *c* is a resulting common boundary node of $\mathcal{R}_2$ and $\mathcal{R}_3$.

Note that the hierarchy needs to be established only once at the time when the vehicle is first introduced in the system. Afterwards, it is dynamically maintained in a fully decentralized manner. Therefore, the storage overhead for tracking the vehicle at each local node is small. HERO automatically reorganizes the hierarchy to control the flooding for location updating to happen mostly in the first few smallest regions. Using flooding for the controlled location updating and hierarchy maintenance is robust and effective when the flooding scale is small [24]. In addition, duplicated useless packets during the flooding are silently dropped which also mitigates the network traffic for location updating. The efficacy of HERO design can be examined more intensively by our extensive simulations.

D. *Protocol Analysis and Parameter Optimization*

By far, a key question remaining unestablished is the configuration of the radii r_i ($1 \le i \le h$) in (1). To conveniently control the maximum number of regions in the hierarchy and to restrain the location updating in small regions close to the vehicle, HERO organizes the hierarchical regions with exponentially increasing sizes.

More specifically, we introduce two protocol parameters: first radius *r* and amplification factor *k*. The radius of the first region is *r* (i.e., r_1=r), and the radius of $\mathcal{R}_i$ is $k^{i-1}r$ (if *k* is an integer). Fig. 2 shows an example with *r* and *k* both equal to 2. More generally, *k* can take any real number greater than one. Since the radius of a region must be an integer in hops, we take the ceiling of $k^{i-1}r$ as the radius of $\mathcal{R}_i$ and further make sure that

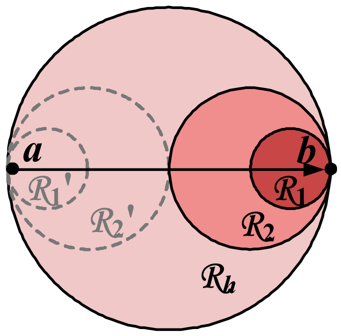

Figure 5. Worst case of location updating, when the vehicle traverses the whole network from node *a* to *b*.

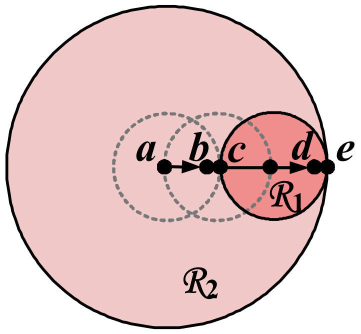

Figure 6. Example of continuous reconstruction of $\mathcal{R}_1$ during the movement from node *a* to *d*.

a region is larger than its immediate inner region. Then the radius of $\mathcal{R}_i$ is defined as,

$$r_1 = r;$$
$$r_i = \begin{cases} \lceil r \cdot k^{i-1} \rceil, \ \textit{if} \ r_{i-1} < \lceil r \cdot k^{i-1} \rceil; \\ r_{i-1} + 2, \ \textit{otherwise}. \end{cases} \qquad (2)$$

With this region organization, the maximum number of hops a query is routed is subject to the following theorem.

Theorem 1. *Given a network with the network diameter (i.e., the maximum hop distance between any pair of nodes) D hops, it takes at most* $\lceil \log_k(D/r) \rceil$ *hops for a query to be answered.*

PROOF. The worst case of a query, where it traverses the maximum number of hops, occurs when the hierarchy is constructed from one end of the network diameter and the query is injected at the other end of the diameter. In this case, according to the definition of the exponential hierarchy, the maximum number of regions contained in the network is $\lceil \log_k(D/r) \rceil$. Since nodes in $\mathcal{R}_1$ always have the latest location information, a query only needs to be routed to a boundary node of $\mathcal{R}_1$. Thereby, a query takes at most $\lceil \log_k(D/r) \rceil - 1$ hops to reach that boundary node. It takes the boundary node one more forwarding hop to finally return the result back to the node that initiates the query. This concludes the proof. □

Despite of the general characteristic of locality, a vehicle prefers to move as straight as possible for a destination. As straight movements can break the maximum number of regions and hence cause the most location updating traffic, we need to investigate this situation and have the following theorem.

Theorem 2. *Suppose that the topology of a network is a disk, the maximum network traffic overhead of location updating for a vehicle moving a distance of D is* $\eta(D) = c(kD^2 + 2r(r-k-1)D - 6r^2)$*, where D is the network diameter and c is a constant coefficient.*

PROOF. Fig. 5 depicts the worst case of location updating among all possible movements with a distance of D, where all constructed regions in the network need to be reconstructed during the movement from node *a* to node *b*. For analysis simplicity, we assume that *k* is an integer. With uniform deployment of local nodes, the network traffic for flooding in $\mathcal{R}_i$ (denoted as S_i) can be approximately evaluated by the area of $\mathcal{R}_i$. Let φ_i denote the updating overhead incurred as a vehicle moves from the boundary of $\mathcal{R}_{i-1}$ to the node immediately next to the boundary of $\mathcal{R}_i$, ($i \geq 2$). For example in Fig. 6, the updating overhead introduced when the vehicle moves from node *a* to node *b* is denoted as ζ_1, and that from node *c* to node *d* is denoted as ζ_2. We have,

$$\begin{cases} \zeta_1 = (r-1) S_1 = c_0 \pi (r-1) r^2 \\ \zeta_i = (k-1) \cdot \left(S_i + \sum_{j=1}^{i-1} \zeta_j \right) \end{cases}, \qquad (3)$$

where c_0 is a constant coefficient. Let ω_m denote the updating overhead incurred as the vehicle traverses the diameter of $\mathcal{R}_m$ from node *a* as shown in Fig. 5. It can be formulated as follows,

$$\begin{aligned} \omega_m &= (2r-1) \cdot S_1 + 2(k-1) \left(\sum_{i=2}^{m} S_i + \sum_{i=1}^{m-1} \sum_{j=1}^{i} \zeta_j \right) \\ &= c_0 \pi \left(2k \cdot r_m^{\ 2} + 2r(r-k-1) r_m - 3r^2 \right) \end{aligned} \qquad (4)$$

Denote $\eta(D)$ as the total updating traffic caused while the vehicle traverses the network, and then $\eta(D) = \omega_h$. Let $c = c_0 \pi / 2$. This concludes the proof. □

Let t_d denote the maximum delay of a query between two neighboring nodes, and t_0 denote the application real-time constraint. We try to minimize the average updating overhead per hop, denoted as $g(r, k)$, under the constraint $\lceil \log_k(D/r) \rceil \leq t_0/t_d$. Let $\log_k(D/r)+1 = t_0/t_d$, then $g(r, k)$ can be reduced as,

$$g(r,k) = c[D^{\frac{t_d}{t_0 - t_d}} \cdot r^{-\frac{t_d}{t_0 - t_d}} (D - 2r) + 2(1 - \frac{3}{D}) r^2 - 2r]. \qquad (5)$$

We develop numerical procedures to compute the approximately optimal value of *r* and *k* respectively based on (5) and the results can be stored into a table. Therefore, HERO supports to track each vehicle with a different real-time constraint by organizing a particular hierarchy for each one with corresponding *r* and *k*. Given a particular set of t_d, t_0 and *D*, the configuration of *r* and *k* can be determined by looking up the table.

V. Design Issues

It is possible that occasionally a vehicle fails to be captured by a RFID reader (e.g., when the vehicle is moving too fast). In addition, a local node may also fail from time to time. This data inaccuracy can be corrected. At any time, a node *n* in region $\mathcal{R}_i$ ($i\geq 2$) should have received an update packet from a boundary node of $\mathcal{R}_{i-1}$ before the node itself captures the vehicle. Otherwise, the corresponding updating process has failed. To solve the problem, node *n* takes the responsibility to trigger the updating for the reorganization of regions from $\mathcal{R}_1$ to $\mathcal{R}_{i-1}$. Unless node *n* itself happens to be a boundary node of $\mathcal{R}_i$, in that case, it performs updating for the reorganization of regions from $\mathcal{R}_1$ to $\mathcal{R}_i$ instead.

As a vehicle keeps moving, it may run out of the current cover of an RFID reader while still has not entered the territories of others. This causes the system have inaccurate vision about the current position of the vehicle before it re-enters into the system. To refine the resolution of tracking accuracy, more RFID readers can be deployed at those places which are more interested to users.

In HERO, a single node failure can be discovered in a short time. An unavailable node can be reported to the system administrator by its neighbors. To join the system for tracking vehicles, a new node (or a recovered node) only needs to contact its neighboring nodes. Then, for each vehicle, the node configures its status to the same as that of the neighbor which resides in the smallest region among all neighbors in the hierarchy. Thereafter, it can perform location updating and query processing properly.

VI. Performance Evaluation

A. *Methodology and Metric Design*

In the simulations, the HERO protocol is implemented using ns2 [5]. Since it is more preferable to leverage cheap ADSL connections than dedicated networks to connect all nodes in the metropolitan-scale system, we use Brite [6] to simulate the underlying Internet topology of Shanghai with 10,000 routers. The overlay network is based on the real complex road network of Shanghai where local nodes are deployed on every crossroad. The size of all used packets is 40 bytes. The typical link transmission delay between two neighbor nodes in the overlay network is 48ms, measured by `ping` between two desktop PCs with 1MB bandwidth ADSL dial-up connections. One of the overlay topologies employed in our simulations is depicted in Fig. 7. The topology containing 1,000 nodes (denoted by small white dots) covers the geographical downtown area of Shanghai. The dark line shows the network diameter in the topology which is 55 hops.

To investigate the impact of the vehicle moving patterns to the HERO design, we use real GPS trace data of taxies which were obtained with GPS technology from August 2006 to October 2006. Taxies can move more randomly and extensively in the whole city and, therefore, have more sense to be considered. A typical trace of a taxi in the downtown area of Shanghai through daytime (on Aug. 13, 2006) is shown in Fig. 7, where red dots are locations captured when the taxi is vacant and the dark dots are those captured while the taxi is delivering passengers. It can be seen that the taxi cruises around within an area most the time when seeking for passengers. This benefits our HERO design best because most of the location updating can be perfectly restricted within small regions. It can also be seen that, when the taxi has a delivery, it runs very fast along the straightest path for its destination. HERO leverages restricted location updating strategy to reduce network traffic while still keeping the whole system up-to-date.

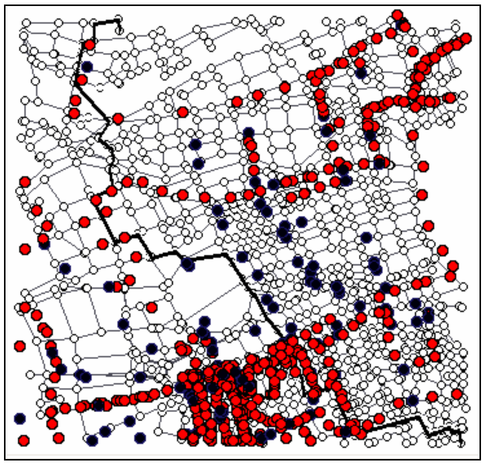
Figure 7. The topology of the downtown area of Shanghai with 1,000 nodes (red dots) deployed at crossroads of this area.

In this section, we compare HERO with several alternative schemes:

- **ST-Updating.** In this scheme, whenever a node captures a vehicle, it updates this information to all other nodes. To reduce the network traffic overhead of this update, the system maintains a global spanning tree. Therefore, only $n-1$ update packets are introduced across the whole network for each update. The strength of this scheme is that each node can answer any query locally, providing minimal query response time.
- **ST-Flooding.** This scheme does not perform vehicle information update in the network and hence no overhead is introduced for location updating. To search for a vehicle, a query is flooded throughout the network. A global spanning tree is used to broadcast the query to reduce the network traffic overhead.
- **Ex-Flooding.** Similar to ST-Flooding but instead of relying on a global spanning tree, it employs expanding flooding. The query is iteratively flooded on the network with ever-increasing scales (by increasing the TTL with 4 hops each time) until the vehicle is found.
- **Random Walks.** This scheme does not perform information update either. To search for a vehicle, five walkers randomly walk on the network. A walker checks with the querying node every 50 steps until either the result has been retrieved or the maximum 2,000 steps have been reached.

We propose two important metrics to evaluate the performance of HERO and the above schemes:

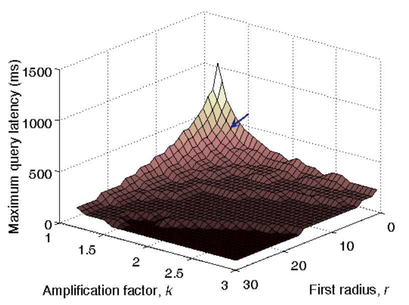

Figure 8. Maximum query latency vs. different protocol parameters.

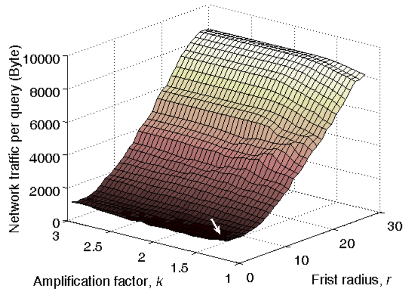

Figure 9. Updating network traffic vs. different protocol parameters.

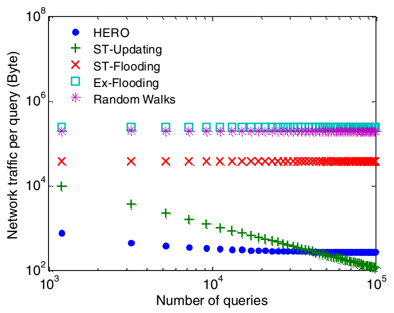

Figure 10. Network traffic per query vs. number of queries.

1) ***Maximum query latency (M_{QL}).*** It refers to the maximum query response time of a successful query. The intention of this metric is to check whether a scheme can guarantee certain real-time requirements.

2) ***Network traffic per query (M_{NT}).*** It can be seen that if there were no query then no location updating would need to be carried out at all. Therefore, to answer a query, the system cost should involve two parts of network traffic, i.e., for location updating and for routing query packets. We investigate the communication cost per query cost by any location updating as well as query processing.

B. *Effects of Protocol Parameters*

We first examine the effects of protocol parameters on the system performance and validate the theoretical analysis. We employ one hour extensive trace data of 100 taxies, randomly generate 10^5 queries for different vehicles during this hour and demand any query should be answered within 500 milliseconds. We vary r from 1 hop to 30 hops with an increment of 1 hop, and vary k from 1.2 to 3 with an increment of 0.05. For each pair of r and k, we repeat the experiment 10 times and present the average.

Fig. 8 and Fig. 9 plot the M_{QL} among all the generated queries and the M_{NT}, respectively, under different configurations of r and k. It shows that the M_{QL} drops dramatically with increasing r and k. The M_{NT} increases as either r or k increases. This is because either a greater r or a greater k leads to a more aggressive updating strategy. In the extreme cases where r equals to D or r equals to one and k equals to D, HERO actually floods every location updating throughout the whole network. In this experiment setting, the numerical computation results of r and k are 2 and 1.393, respectively. The arrows in Fig. 8 and Fig. 9 show their corresponding positions. It is clear to see that, with this configuration of r and k, HERO can answer any query within the real-time constraint with the M_{NT} approaching to the practical minimum.

C. *Impact of Query Quantity*

In this experiment, we investigate the impact of the query quantity on the system performance. We take the same setting as the previous experiment with r and k set to 2 and 1.393, respectively. We vary the total number of queries from 10^5 to 10^7 with increments of 400.

Among all queries, the M_{QL} of HERO is 480ms which is strictly shorter than the real-time constraint. In ST-Updating, the M_{QL} is about 14ms which is for local database operations. The other schemes cannot guarantee to satisfy the real-time requirement. The M_{QL} of ST-Flooding and Ex-Flooding is 5,232ms and 14,120ms, respectively. The M_{QL} of Random walk is about 10^5ms due to the search step limitation of 2,000. Fig. 10 plots the M_{NT} with different numbers of queries per vehicle. The M_{NT} of HERO is much less than that of other schemes. In addition, it declines as the number of queries increases. It can be seen that, with this experiment setting, HERO has less query overhead than ST-Updating until the number of queries for the same vehicle exceeds 41,400. This is very interesting to find that the number of queries decides whether ST-Updating or HERO is preferable. However, we argue that it is impractical that a single vehicle would be queried so tensely within one hour in a region with 1,000 nodes.

VII. Conclusion and Future Work

In this paper we have presented the real-time tracking protocol HERO for the metropolitan-scale intelligent transportation system. Exploiting the locality of vehicle movements in the urban area, HERO adaptively updates the locations of a vehicle according to the innovative hierarchical structure. HERO significantly reduces network traffic while still satisfying the real-time requirement. As a fully distributed protocol, this protocol is highly scalable to the number of users, the number of vehicles and the system scale as well. Comprehensive simulations based on the real road network and trace data of vehicle movements demonstrate the efficacy of HERO.

This is an on-going research and system effort in tracking various vehicles in the metropolitan-scale system. Following the current work, we have a lot of more exciting yet challenging topics ahead. One of these topics is the privacy implications of tracking personal vehicles all the time. The government will guarantee to protect individual privacy by authorizing legal individuals and corporations with different privileges to access appropriate vehicles. Next, we will delve into designing better location updating schemes such that update overhead can be reduced as much as possible. A realistic prototype testbed will be built to validate our design and study its performance under real complex environments. Improvements will be made based on the realistic studies before it comes to be deployed in the large-scale SG system.

REFERENCES

[1] Shanghai City Comprehensive Transportation Planning Institute, http://www.scctpi.gov.cn/chn/chn.asp, 2006.

[2] LoJack Corp., “Stolen Vehicle Recovery System,” http://www.lo-jack.com/what/stolen-vehicle-recovery-system.cfm, 2007.

[3] Ltd Shanghai Super Electronic Technology Co., http://www.super-rfid.net/english/, 2007.

[4] “Transportation Recall Enhancement, Accountability, and Documentation (TREAD) Act,” the 106th United States Congress, 2000, http://www.citizen.org/documents/TREAD%20Act.pdf.

[5] The Network Simulator, http://www.isi.edu/nsnam/ns/, 2006.

[6] BRITE, http://www.cs.bu.edu/brite/, 2006.

[7] “The Gnutella Protocol Specification V0.6,” http://rfc-gnutella.sourceforge.net, 2005.

[8] Hongzi Zhu, Yanmin Zhu, Minglu Li, and Lionel M. Li, “ANTS: Efficient Vehicle Locating Based on Ant Search in Shanghai Tranportation Grid”, in Proceedings of ICPP, 2007.

[9] A. Bakker, E. Amade, G. Ballintijn, I. Kuz, P. Verkaik, I. van der Wijk, M. van Steen, and A.S. Tanenbaum, “The Globe Distribution Network,” in Proceedings of USENIX Annual Conference, 2000.

[10] Alminas Civilis, Christian S. Jensen, and Stardas Pakalnis, "Techniques for Efficient Road-Network-Based Tracking of Moving Objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 698-712, 2005.

[11] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis, “Novel Approaches to the Indexing of Moving Object Trajectories,” in Proceedings of VLDB, 2000.

[12] George Kollios, Dimitrios Gunopulos, Vassilis Tsotras, Alex Delis, and Marios Hadjieleftheriou, "Indexing Animated Objects Using Spatiotemporal Access Methods," IEEE Transactions on Knowledge and Data Engineering, vol. 13, pp. 758-777, 2001.

[13] Dan Lin, Christian S. Jensen, Beng Chin Ooi, and Simonas Saltenis, "Efficient Indexing of the Historical, Present, and Future Positions of Moving Objects," in Proceedings of MDM, 2005.

[14] Mindaugas Pelanis, Simonas Saltenis, and Christian S. Jensen, “Indexing the Past, Present, and Anticipated Future Positions of Moving Objects,” ACM Transactions on Database Systems, vol. 31, pp. 255-298, 2006.

[15] John F. Roddick, Max J. Egenhofer, Erik Hoel, and Dimitris Papadias, "Spatial, Temporal and Spatio-Temporal Databases - Hot Issues and Directions for Phd Research," in Proceedings of SIGMOD, 2004.

[16] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," University of California at Berkeley, Technical Report, UCB/CSD-01-1141, 2001.

[17] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” in Proceedings of ACM SIGCOMM, 2001.

[18] Antony Rowstron and Peter Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in Proceedings of IFIP/ACM Int. Conf. Distributed Systems Platforms, 2001.

[19] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, “A Scalable Content-Addressable Network,” in Proceedings of ACM SIGCOMM, 2001.

[20] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker, “Search and Replication in Unstructured Peer-to-Peer Networks,” in Proceedings of the 16th international conference on Supercomputing, 2002.

[21] Christos Gkantsidis, Milena Mihail, and Amin Saberi, "Random Walks in Peer-to-Peer Networks," in Proceedings of IEEE INFOCOM, 2004.

[22] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Davavrat Shah, “Gossip Algorithms: Design, Analysis, and Applications,” in Proceedings of IEEE INFOCOM, 2005.

[23] David Kempe, Alin Dobra, and Johannes Gehrke, "Gossip-Based Computation of Aggregation Information," in Proceedings of IEEE FOCS, 2003.

[24] Song Jiang, Lei Guo, and Xiaodong Zhang, “LightFlood:an Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems”, In:Proceedings of ICPP, 2003.