ROADTRACK: Scaling Location Updates for Mobile Clients on Road Networks with Query Awareness

Peter Pesti, Ling Liu, Bhuvan Bamba, Arun Iyengar*, Matt Weber
College of Computing, Georgia Institute of Technology, Atlanta, USA
{pesti,lingliu,bhuvan,mattweb}@cc.gatech.edu
*IBM Research, T.J. Watson Research Center, Yorktown Heights, USA
*aruni@us.ibm.com

ABSTRACT

Mobile commerce and location based services (LBS) are some of the fastest growing IT industries in the last five years. Location update of mobile clients is a fundamental capability in mobile commerce and all types of LBS. Higher update frequency leads to higher accuracy, but incurs unacceptably high cost of location management at the location servers. We propose ROADTRACK – a road-network based, query-aware location update framework with two unique features. First, we introduce the concept of precincts to control the granularity of location update resolution for mobile clients that are not of interest to any active location query services. Second, we define query encounter points for mobile objects that are targets of active location query services, and utilize these encounter points to define the adequate location update schedule for each mobile. The ROADTRACK framework offers three unique advantages. First, encounter points as a fundamental query awareness mechanism enable us to control and differentiate location update strategies for mobile clients in the vicinity of active location queries, while meeting the needs of location query evaluation. Second, we employ system-defined precincts to manage the desired spatial resolution of location updates for different mobile clients and to control the scope of query awareness to be capitalized by a location update strategy. Third, our road-network based check-free interval optimization further enhances the effectiveness of the ROADTRACK query-aware location update scheduling algorithm. This optimization provides significant cost reduction for location update management at both mobile clients and location servers. We evaluate the ROADTRACK location update approach using a real world road-network based mobility simulator. Our experimental results demonstrate that the ROADTRACK query aware location update approach outperforms existing representative location update strategies in terms of both client energy efficiency and server processing load.

1. INTRODUCTION

We are entering a wireless and mobile Internet era where people and vehicles are connected at all times. In the past five years we have witnessed an astonishing growth of mobile commerce and location based applications and services, which not only extend many traditional businesses into new product offerings (e.g., location based advertisement, location based entertainment) but also create many opportunities for new businesses and innovations. Consider a metropolitan area with hundreds of thousands of vehicles. Drivers and passengers in these vehicles are interested in information relevant to their trips. For example, some driver would like her vehicle to continuously display on a map the list of Starbucks coffee shops within 10 miles of her current location. Another driver may want to monitor the traffic conditions five miles ahead of its current location (e.g., traffic flow speed). The challenge is how to effectively monitor the location updates of mobile users and continuously serve location queries (traffic conditions, parking spaces, Starbucks coffee shops) with an acceptable delay, overhead, and accuracy, as the mobile users move on the road.

There are two key performance challenges that may affect the system scalability and service quality in future mobile systems supporting location-dependent services and applications: (1) the high cost of network bandwidth and energy consumed on the mobile clients for frequent location tracking and updates at the location servers; and (2) the challenge of scaling large amount of location updates at the location server as the number of mobile clients demanding to be tracked increases in a location determination system. Furthermore, handling frequent load peaks at location update synchronization points is also a challenge, since the server has to simultaneously handle location updates from a large number of mobile clients, and re-evaluate all registered spatial location query services.

Location Update Problems and Existing Approaches
Monitoring location updates and evaluation of location queries over static and moving objects upon location updates have become the necessity for many mobile systems and location-based applications, such as fleet management, cargo tracking, child care, and location-based advertisement and entertainment. Frequent updates cause high update processing cost at the location server and high power consumption at the mobile clients [1]. Several European mobile service providers have started the cost-based location management for mobile object tracking. For instance, different pricing models are applied to high frequency location updates at different time intervals, such as every three minutes, every one minute, every 30 seconds, and so forth.

In contrast to location determination systems where localization techniques are employed to determine the position of a mobile subscriber within the area serviced by the wireless network, the location update management addresses the problem of when and where to update the locations of mobile subscribers currently hosted in the system. Representative location update strategies to date include periodic update (time based scheme), point-based up-

Proceedings of the VLDB Endowment, Vol. 3, No. 2
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

date using dead-reckoning, velocity vector based update, and segment based updates [2]. However, existing location update strategies are inefficient because i) they are common to all mobile users, and ii) they assume that location updates of mobile clients are autonomous and all mobile users should manage their location updates using a uniform strategy. To the best of our knowledge, no customization or differentiation is incorporated to the design of location update management strategies.

We argue that, as mobile and hand-held devices become more pervasive, more capable, and both GPS and WiFi enabled [3, 4], as the operation cost of location update management continues to grow, these assumptions are no longer realistic. For instance, most of the mobile systems and applications today need to manage a large and evolving number of mobile objects. Often, only a subset of mobile objects is of interest to registered location query services. Thus, tracking location updates of all mobile clients uniformly is no longer a cost effective solution. It is obvious that the location update strategy for those clients that are of no interest to any nearby and active location query services should be different from and less costly compared to the location update strategy designed for mobile objects that are the targets of active location query services in the system.

Motivated by these observations, in this paper we present ROADTRACK – a road-network based, query-aware location update framework by introducing precincts and encounter points as two basic techniques to confine location updates to the need of existing location query services. These two basic building blocks enable us to effectively differentiate and manage location updates for mobile objects traveling on road networks. We utilize precincts to manage the spatial resolution of location updates for mobile clients that are not immediate targets of any existing location query services. We introduce encounter points to implement the query-aware location update strategy for mobile clients nearby active location queries. By combining precincts and encounter points, we can balance the benefit and cost of query awareness and speed up the computation of encounter points. The ROADTRACK location update management offers three unique advantages. First, encounter points as a fundamental query awareness mechanism enable us to control and differentiate location update strategies for mobile clients in the vicinity of active location queries from the rest. Second, by employing system-defined precincts, we can effectively manage the desired spatial resolution of location updates for mobile clients with different needs for query awareness. Third but not the least, we improve the efficiency of ROADTRACK location update approach by employing a suite of road-network based check-free interval optimization techniques. We evaluate the ROADTRACK approach to location update management based on a real world road-network mobility simulator [5]. Our experimental results show that by making location update management *query aware*, ROADTRACK approach significantly outperforms existing representative location update strategies in terms of both client energy efficiency and server processing load.

The rest of the paper is organized as follows: We outline the reference system model and discuss the design philosophy through an analysis of existing representative location update strategies in Section 2. In Section 3, we introduce the concept, the computation, and the usage of encounter points and the precinct and encounter based location update strategy, including the data structure used at both the server and the client side. We present the encounter points based check-free interval optimization in Section 4. Section 5 reports our experimental evaluation on the effectiveness of our ROADTRACK query aware location update approach. We conclude the paper with related work and a summary of contributions.

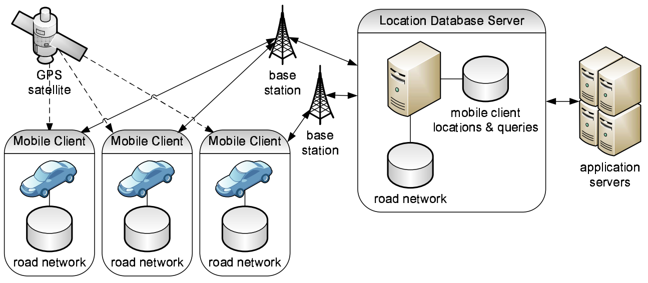


Figure 1: Overview of the system architecture

2. SYSTEM OVERVIEW

A location update and monitoring system typically consists of a location database server, some base-stations, application servers, and a large number of mobile objects (mobile clients) and static objects (such as gas stations, restaurants, and so on). The location database server (location server for short) manages the locations of the moving objects. The application servers register location queries of interest, and synchronize with the location server to continuously evaluate the queries against location updates.

Figure 1 gives an architectural overview of the reference location monitoring system used in the context of ROADTRACK development. We assume that mobile clients and the location server have a local copy of the same road network database that constrains the movement of the clients; clients may store this on an SD card. For the clients with limited storage, a tile based partitioning of the road network map can be used [6]. We assume that the mobile clients are able to communicate with the server through wireless data channel, and they have computing capabilities to run our light-weighted road network locator, which uses a static R-tree index on road segments to find their own road network locations based on their GPS positions through map matching. Mobile clients may also obtain their positions from the location determination system they subscribe to, such as Google's locator service available on iPhone and other hand-held devices.

2.1 Road network model

The road network is represented by a single undirected graph $G = (\mathcal{V}, \mathcal{E})$, composed of the junction nodes $\mathcal{V} = \{n_0, n_1, \ldots, n_N\}$ and undirected edges $\mathcal{E} = \{n_i n_j | n_i, n_j \in \mathcal{V}\}$. In this paper we frequently refer to an edge $n_i n_j$ as a road segment connecting the two end nodes n_i and n_j. The listing order of the two end nodes of a segment $n_i n_j$ serves as the basis to determine the direction of the *progress* coordinate axis from node n_i to node n_j along the segment $n_i n_j$. In other words, the segment $n_i n_j$ runs from $p = 0$ at the first listed node (n_i) to $p = length(n_i n_j)$ at the second listed node (n_j). Though in this paper we model the road network using undirected graphs for simplicity, our methods can be extended to directed graphs. Junction nodes have either two or more connecting road segments, or are dead-end nodes with only one connecting road segment. A *road network location*, denoted by $L = (n_i n_j, p)$, is a tuple of two elements: a road network segment $n_i n_j$ and the *progress* p along the segment. The road network distance is used as the distance metric in our system. The distance between two locations $L_1 = (n_{i_0} n_{i_1}, p_1)$ and $L_2 = (n_{i_k} n_{i_{k+1}}, p_2)$ is the length of the shortest path between the two positions L_1 and L_2, formally defined as follows:

$$dist(L_1, L_2) = length(n_{i_0} n_{i_1}) - p_1 + p_2 + \min_{\{i_1, i_2, \ldots, i_k\}} \sum_{\alpha=1}^{k-1} length(n_{i_\alpha} n_{i_{\alpha+1}}).$$

2.2 Design Guidelines

A number of positioning systems are made publicly available for tracking the location update of mobile objects moving on the road network, such as Google's Latitude and Skyhook wireless WiFi positioning system [3]. Frequent location updates enable the location server to keep track of mobile clients' current locations and ensure the accuracy of the location query results. The algorithm that mobile clients employ to determine when and where to update their locations is often referred to as the location update strategy. We below describe the motivation, the advantages, and the challenges of our query-aware location update framework by analyzing and comparing a number of representative location update strategies.

Periodic update strategy. A *periodic update strategy* is the simplest time-based location update strategy, in which the location server maintains the location update for each mobile client at a fixed time interval. This update strategy implies that mobile clients are treated as stationary between updates.

Point-based update strategy. This approach uses the distance-based scheme and the server only record an update when the mobile client travels more than a delta threshold away in distance from the location of last update. The number of location updates per unit time will depend upon the speed of the mobile user.

Vector-based update strategy. A *vector based update strategy* uses the velocity vector of the mobile client to make a simple prediction about its location. An update is only sent when the current location of the mobile client deviates from its predicted location by an amount that is larger than a system-defined delta distance threshold. This strategy treats the velocity vector of the client as constant between updates.

Segment based update strategy. A *segment based update strategy* utilizes the underlying road network to limit the number of updates. Mobile clients are assumed to move at a constant speed on their current road segment. An update is sent when the distance between the current and the predicted location is larger than a system-defined delta threshold. We assume that mobile clients change their velocities at the end of each segment, i.e., the mobile client is assumed to have stopped at the segment end node and can change its movement speed and direction and move forward accordingly. Thus an update will be sent when the mobile client departs from a segment end node by delta distance. We refer the reader to [2] for more on these strategies.

Motivation of Our Approach.
We have discussed four representative location update strategies and each of them has some weakness in terms of both client energy-efficiency and network bandwidth or server load optimization. Furthermore they all suffer from the common inefficiency − the location update decision of mobile clients is independent of whether there are any location query requests nearby. It is obvious that when mobile clients travel in a region where there are no location queries, one can benefit by using a location update strategy that enable the location server to record their location updates at some critical location points, leading to significant saving in terms of client energy and bandwidth consumption as well as server load reduction. In ROADTRACK two criteria are used to determine what should be considered as critical location update points. First, we need to increase the location query awareness of mobile clients. By making mobile users aware of queries in their vicinity, one can avoid making those superfluous updates. Second, we need to maintain certain freshness of location updates for those mobile clients that are not in the vicinity of any location queries to maintain adequate location tracking capability of the system. The second criterion ensures that all mobile clients need to update their current location at the location server from time to time in order to keep their location record update to date at the location server, though different mobile clients may use different scale of location resolution.

Bearing these two design guidelines in mind, we develop a *query-aware, precinct based update strategy*. Concretely, we introduce the concept of encounter point and the concept of precinct as two building blocks. By keeping track of the encounter points for each mobile client moving on the road network, we are able to use the query awareness to differentiate the location update strategy used for mobile clients that are in the vicinity of active queries from the location update strategy used for the mobile clients that are not targets of any location queries. The use of precincts constrains the set of encounter points that a mobile client needs to keep track of to be small, and sets an upper bound on when the mobile clients have to update their locations regardless of whether there are location queries nearby. To further reduce the cost of checking whether a mobile is close to the border points of its current precinct or one of its encounter points, we develop a road network distance based check-free interval optimization, providing significant reduction in terms of the number of wakeups at the mobile client and the server update load.

The ROADTRACK query aware location update strategy is applicable to all moving objects in a road network setting, be it vehicles or pedestrians. This research is based on the assumption that all moving objects are either moving on the public road networks, or walk paths such as indoor buildings or university campus walk paths. As long as these walk paths can be modeled as graphs, our approach can be applied directly.

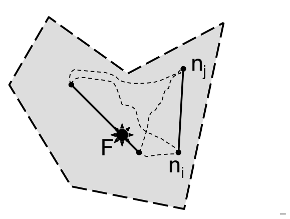
3. PRECINCT BASED UPDATE STRATEGY

In this section we describe the basic design of our precinct and encounter point based location update method, and defer the check-free interval based optimization to the next section.

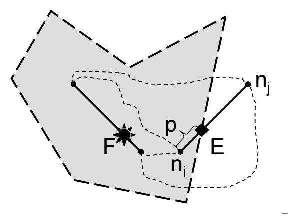
3.1 Precinct and Encounter Point

Precinct.
Precinct is introduced in ROADTRACK for dual purposes. First, every mobile object is associated with a precinct in which it currently resides. We use precinct as the spatial upper bound to enforce location updates of all mobiles when they cross their current precinct boundary and enter a new neighbor precinct. Second, we employ precinct to limit the scope of query awareness and balance the tradeoff between the level of location accuracy maintained at the server and the reduction of location update cost at the server. For example, queries about the restaurants in Miami are far away from the current location of a mobile client traveling in Atlanta downtown. Thus, the mobile clients in Atlanta downtown should not be made aware of queries about restaurants in Miami. By introducing system-defined *precincts*, we can conveniently limit the scope of query awareness for mobile clients residing within their precincts. This also ensures that the number of encounter points maintained at a mobile client is small.

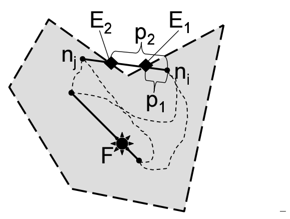
A precinct $P = \{\mathcal{V}_P, \mathcal{E}_P\}$ is a subgraph of the road network $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}_P \subset \mathcal{V}$ and $\mathcal{E}_P \subset \mathcal{E}$. Nodes in $\mathcal{V}_P$ are either *internal* or *border* nodes. Each internal node is reachable from all other nodes of a precinct on a path composed of only internal nodes. All edges in $\mathcal{E}$ that are connected to an internal node in



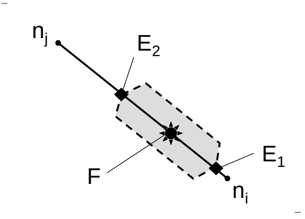
(a) n_in_j completely covered



(b) n_in_j partially covered



(c) n_jn_i partially covered near both ends (two encounter points)



(d) n_jn_i partially covered in the center (two encounter points)

Figure 2: Four major cases for determining encounter points on the segment with end-nodes n_i and n_j. The shaded coverage area represents the query region of query $Q(R, F)$ computed R road network distance away from focal location F.

$\mathcal{V}_P$ are also in $\mathcal{E}_P$. The partitioning of the road network graph is created during the system initialization, and is stored together with the road network data maintained at both the server and the mobile clients. We present the precinct construction algorithms in the next section.

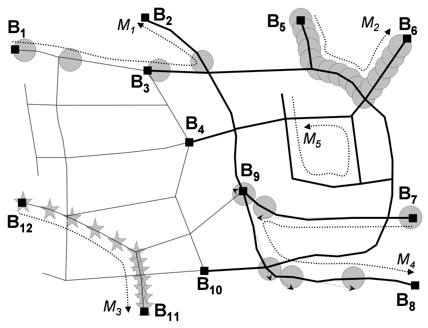
Encounter Points.
We first informally introduce the concept of encounter point. Let $P = \{\mathcal{V}_P, \mathcal{E}_P\}$ denote a precinct and $Q(R, F)$ denote an active location query, where R is the query radius in road network distance and F is the focal location of Q represented using the road network location defined in Section 2. The query Q is said to be relevant to the precinct P if a segment $n_in_j \in \mathcal{E}_P$ is entirely included in the query region R as shown in Figure 2(a) or partially covered by the query region R. Assume that the shaded area in Figure 2 represents the query region computed in terms of road network distance from the focal location of the query, e.g., the query range of 2 miles from the focal location F. If a segment crosses the query boundary, i.e., one end-node is inside the query region and the other end-node is outside R, then we say that the segment is partially covered by the query. We call the road network location where a partially covered segment crosses the query boundary an *encounter point*. Figure 2(b) shows an example encounter point E. It is important to note that even if both end-nodes are inside the query region, the segment may only be partially covered, if there exists a network location L on the segment whose distance to F is greater than the query range specified, i.e., $\exists L|dist(F, L) > R$. In this case there are two encounter points for the query on a single segment (see Figure 2(c)). When the query range is small, it is possible that the query only covers a portion of the segment on which the query focal location F resides, thus there are two encounter points on a single segment but with both end-nodes outside the query region (Figure 2(d)).

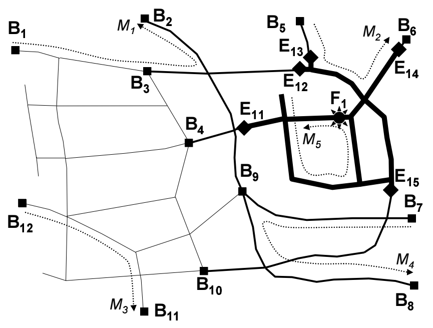
Formally, given a set of location queries (Q) over the road network $G = (\mathcal{V}, \mathcal{E})$, one can determine the set of encounter points $\mathbb{E}_F = \{E_1, \ldots, E_n\}$, each of which (E_j) is associated with a range query $Q_i(R_i, F_i)$ with focal location F_i and range R_i, and is represented as a road network location that is exactly R_i distance from F_i. In other words, the set of encounter points $\mathbb{E}$ satisfies that $\forall E_i \in \mathbb{E}_F, \exists Q_i(R_i, F_i)$ such that $dist(F_i, E_i) = R_i$ and $\nexists L|dist(F_i, L) = R \wedge L \notin \mathbb{E}_F$, i.e., every encounter point is a road network location that is exactly range R_i distance from F_i. The encounter points are defined on the road network. When a mobile client meets or crosses an encounter point, it indicates that the client exits or enters the scope in which the query result is computed. Therefore, we use the encounter points as the critical location reference points for those mobile clients to update their locations at the server whenever they encounter these critical points on the move.

Comparison with existing update strategies.
In Figure 3(a) we show five mobile clients traveling on a portion of a road network, each following a distinct update strategy. The two precincts (west and east) have the common border points B_3, B_4, B_9, B_{10}, and connect to the rest of the road network at all the other border points (all border points shown as black squares). M_1 (upper left) is doing segment-based updates, triggering updates each time the client departs a segment end-node by $delta$ distance. The grey circles show the delta-radius circles around the mobile's location when the updates occur. M_2 (upper right) has a point-based update strategy, and thus sends an update whenever its current location is at least $delta$ distance from its last reported location. M_3 (lower left) is a periodic update mobile client, updating every t seconds. The mobile initially travels fast, continuing at a slow pace; as a result, updates may be spatially too sparse initially, and too dense when speeds are low. We show the locations at the time of updates as stars, since – unlike for M_1, M_2 and M_4 – there is no distance threshold for periodic updates. M_4 (lower right) has a vector-based update strategy, and consequently segment geometry along the trajectory is the primary determinant of update scheduling. However, all these mobiles' updates are wasted, as there are no outstanding queries on this portion of the road network. The fifth mobile client, M_5, following a RoadTrack update strategy, sends no updates, as there are no queries present, and its trajectory does not cross any precinct boundary points.

In Figure 3(b) a range query with focal location F_1 (sun symbol) is installed, with the associated encounter points $E_{11} \ldots E_{15}$ (black rhombus symbol). Note that dead-ends are not E points inside a query coverage area (and not B points inside a precinct). We now ask all mobiles to follow a RoadTrack strategy: M_1 and M_3 cross and update on precinct boundary points only $(B_1, B_2, B_3$; and $B_{12}, B_{11})$. M_2 enters, then exits the query region, and thus also updates on encounter points $(B_5, E_{13}, E_{14}, B_6)$. M_4 crosses boundary point B_9, but remains in the same precinct, and thus only updates on B_7, B_8. Note that B_9 is a real boundary point, as not all connected segments are in the same precinct, and thus a precinct crossing is possible; whether this occurs or not is not known in advance, so it is imperative for M_4 to consider B_9 as a potential update trigger. Finally, M_5 sends no updates, as it does not cross any B or E points. Note that being on the inside or outside of a query region makes little difference to mobile clients: after the initial query evaluation (during query insertion), neither client activity completely outside, nor completely inside the query coverage area changes the query result. Furthermore, as precincts are used to scope query awareness, mobiles in the west precinct (e.g. M_3) need not even consider the query's encounter points (which are all in the east precinct).

In Figure 3(c) an additional range query is added, in the west precinct. M_1 now also updates on this new query region's encounter points (E_{21}, E_{22}), but after entering the east precinct via B_3 it no longer needs to consider any points inside the east precinct.


(a) Updates without query-awareness


(b) Single query (F_1)

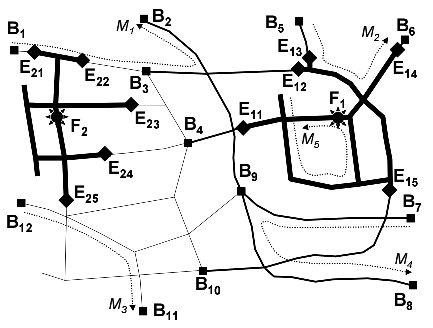

(c) Two queries (F_1, F_2)

Figure 3: Example scenario with encounter points (E) and precinct border points (B) as update trigger points

3.2 Construction of Precincts

Clearly the entire road network is a legitimate precinct. Similarly, the other extreme is the single-segment precinct, where each segment of the road network is considered as one precinct. We can use road network distance or hop count to define the size of the preferred precincts. Assume that we use a system defined network distance threshold to partition the road network into precincts. The algorithm for constructing precincts is similar to a network expansion algorithm. A precinct is constructed by starting at the chosen segment and expanding along the neighboring segments and computing the network distance. This process repeats until the network distance threshold is reached. The construction process is repeated on the remaining segments until all segments in the road network are grouped into precinct-based partitions. A distance-metric based partitioning uses $dist(n_c, n_k) = dist(n_c, n_j) + length(n_j n_k)$ for distance expansion. The algorithm for constructing the precinct partition of a given road network proceeds in three steps. (1) The partition algorithm starts by marking all segments and all junctions as 'uncovered'. (2) A precinct center node n_c is selected at from an ordered queue of uncovered nodes (we elaborate on this ordering below). A queue is maintained during the precinct construction process, which contains a list of candidate nodes in ascending order of their distance from n_c. A node in the road network is a candidate node for the precinct centered at n_c if its distance to n_c is within the system supplied distance threshold. The queue initially contains only n_c. At each expansion step, the entry $(n_j, dist(n_c, n_j))$ at the head of the queue is removed, n_j is marked as 'internal', and all uncovered segments connected to n_j are added to the list of segments covered by the precinct. For segment $n_j n_k$, n_k is added to the list of nodes covered by the precinct, and this node's distance is calculated by $dist(n_c, n_k) = dist(n_c, n_j) + length(n_j n_k)$. If n_k is marked as 'border' (for some other precinct), then it is added to the list of nodes covered by the current precinct with a 'border' flag; otherwise, n_k is marked as 'internal' and $(n_k, dist(n_c, n_k))$ is added to the queue, unless a $(n_k, dist(n_c, n_k)')$ is already in the queue with $dist(n_c, n_k) \geq dist(n_c, n_k)'$. When the distance of the queue head node is larger than the specified precinct range, the precinct construction is concluded by marking all remaining nodes in the queue as 'border', and adding them to the list of nodes covered by the current precinct with the 'border' flag. (3) The algorithm continues with the creation of the next precinct until there are no uncovered nodes. When no uncovered nodes remain, there may still be uncovered segments, whose both end-nodes are border-points for other precincts. Single-segment precincts are constructed for each of these remaining uncovered segments.

An alternative approach to constructing precincts is to use the segment count (or hop count) metric, i.e. we use $dist(n_c, n_k) = dist(n_c, n_j)+1$. Figure 4 shows a partitioning of an example graph with both methods. The randomly selected precinct center nodes are marked by n_1, n_2, n_3 in both cases and are selected in the order of node index. Border nodes are shown with a solid square. Single-segment precincts are highlighted with a grey background. Both hop-count based partitioning (left in Figure 4) and the distance based partitioning (right in Figure 4) shows five precincts: three precincts centered by n_1, n_2, n_3 respectively and two single-segment precincts.

As we mentioned, nodes are selected to serve as precinct centers according to a pre-specified ordering. The ordering method has no bearing on the correctness or utility of the precincts, but may have implications for both the number of client wakeups and the number of updates received by the server. As a result, we can use a random seeding of precincts as our baseline scenario. Instead of such a naïve approach, a node ordering heuristic may be applied, whereby the algorithm prioritizes nodes that lie on many fast roads, as such nodes are likely to be important traffic junctions. This means that we score nodes by the sum of speed limits of their connecting segments, and always choose an uncovered node with the highest score as the next precinct center. In formulating this heuristic, our expectation is that if mobile clients take the shortest path to their destinations, high-speed roads and junctions will see more traffic than low-speed ones. Then, as we place junctions with high potential throughput in precinct centers, high-traffic portions of the road network are covered with relatively fewer precincts, and thus have the prospect of saving some border-point triggered updates and allowing longer check-free intervals between client wakeups.

Let deg denote the average degree of a node. With h-hop based partitioning, the average number of nodes in a precinct may be estimated as:

$$|\mathcal{V}_P|_{avg} \approx 1 + deg \cdot \sum_{i=0}^{h-1} (deg - 1)^i,$$

and the average total length of the segments in a single precinct is calculated by

$$Len_P \approx \sum_{i=1}^{h} l \cdot deg^i = \frac{l(deg - deg^{h+1})}{1 - deg}.$$

With d-distance based partitioning, we can substitute $h = \frac{d}{l}$ above, where l is the average length of a road network segment.

$|\mathcal{V}_P|_{avg}$ is independent of the size of the complete road network. For each precinct, distances between all nodes are pre-computed

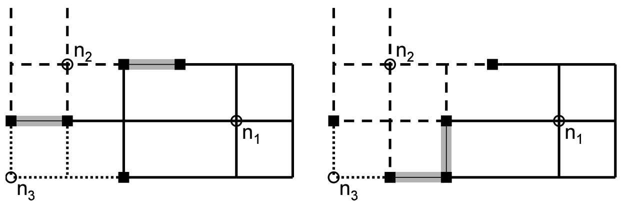


Figure 4: Graph partitioning with h=2 hop-based (left) and r=2 km distance-based (right) algorithm

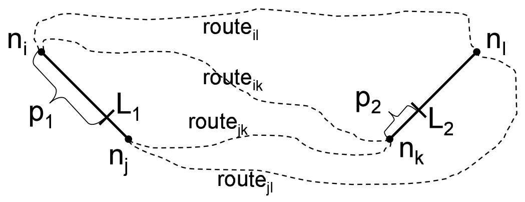


Figure 5: O(1) computation of distances between two arbitrary road network locations L_1 and L_2.

using the Floyd-Warshall algorithm and stored as a D distance matrix for this precinct. The complexity of this step for all precincts is $\frac{|\mathcal{V}|}{|\mathcal{V}_P|_{avg}} \cdot O(|\mathcal{V}_P|^3_{avg}) = O(|\mathcal{V}| \cdot |\mathcal{V}_P|^2_{avg}) = O(|\mathcal{V}|)$. Thus, given a road network $G = (\mathcal{V}, \mathcal{E})$ and its precinct partition $P = \{\mathcal{V}_P, \mathcal{E}_P\}$, the total storage space for the D distance matrices require $\frac{|\mathcal{V}|}{|\mathcal{V}_P|_{avg}} \cdot O(|\mathcal{V}_P|^2_{avg}) = O(|\mathcal{V}| \cdot |\mathcal{V}_P|_{avg}) = O(|\mathcal{V}|)$ storage space. The distance between an arbitrary location $L = (n_i n_j, p)$ and node n can be computed using the node-to-node distances from n to the two end-nodes (n_i and n_j) of the segment $n_i n_j$ that L lies on: $dist(L, n) = min(dist(n_i, n) + p, dist(n_j, n) + (length(n_i n_j) - p))$. The distance between any two locations $L_1 = (n_i n_j, p_1)$ and $L_2 = (n_k n_l, p_2)$ can be computed as the minimum of the lengths of four potential routes as follows (see Figure 5):

$$\begin{aligned}
&dist(L_1, L_2) = \\
&\quad = min(route_{ik}, route_{il}, route_{jk}, route_{jl}) \\
&\quad = min(\, dist(n_i, n_k) + p_1 + p_2, \\
&\qquad dist(n_i, n_l) + p_1 + (length(n_k n_l) - p_2), \\
&\qquad dist(n_j, n_k) + (length(n_i n_j) - p_1) + p_2, \\
&\qquad dist(n_j, n_l) + (length(n_i n_j) - p_1) \\
&\qquad\quad - (length(n_k n_l) - p_2)\,).
\end{aligned}$$

3.3 Data structures

In this section we give a brief overview of the data structures used at the server-side and the client-side to facilitate the understanding of our precinct based location update framework.

Server side data structures.

Node Table, NT = (nid, $\{sid\}$) stores road network nodes with the sid segment identifiers for the segments that connect to the node. A hash table index on the nid node identifiers allows constant speed lookup.

Segment Table, ST = (sid, nid_1, nid_2, pid, $\{oid\}$, $\{qid\}$) stores road network segments with the two end-nodes (nid_1 and nid_2). A hash table on the sid segment identifiers allows constant speed lookup. We store the identifier of the precinct covering the segment (pid), the client identifiers for clients on the segment ($\{oid\}$), and the list of query identifiers for queries (fully or partially) covering the segment ($\{qid\}$).

Precinct Table, PT = (pid, $\{sid\}$, $\{(nid, isBorder)\}$, D) stores information about a precinct with the identifier pid, along with the list of road network segments covered ($\{sid\}$), the list of nodes covered along with a flag showing whether the node is 'border' or 'internal' ($\{(nid, isBorder)\}$), and the pre-computed node-to-node distance table (D).

Query Table, QT = (qid, oid, range, F, $\{(sid, E, dir)\}$, $\{result\}$) stores queries in the system with the qid query identifier, the oid identifier of the client the query is attached to, the $range$ specifying the road network distance based range of the query, and F giving the focal location of the query. The $\{(sid, E, dir)\}$ list contains tuples of segment identifiers of segments at least partially covered by the query, encounter point locations for segments not fully covered (or $null$ for a completely covered segment), and a flag indicating which part of the segment is inside the query region (source-side or target-side). The $\{result\}$ list stores client identifiers for the clients that satisfy the query.

Client Table, CT = (oid, L, M) stores information about mobile clients in the system. The table is indexed on the client identifier attribute oid. L is the most recently updated road network location of the client, stored as a (sid, p) tuple, comprising of the *sid* segment identifier and the *p* progress. The M provides the client's mobility features required by the system, such as movement speed, trajectory, and so forth.

Client side data structures

NT, ST, and PT are also present on the client side as part of their map database.

Current Encounter points Table, CET = (sid, $\mathbb{E}$, dir) contains the encounter points found for all queries in the client's current precinct. Each mobile client only stores the encounter points for the precinct that includes the segment on which it is located. The CET is delivered to the client by the server when a client informs the server that she enters a new precinct. Also the CET at a client is incrementally updated by the server to reflect query insertions or deletions.

3.4 Computing with Encounter Points

Encounter points need to be computed whenever a new query is inserted into the system, or an existing query is terminated and removed from the system.

Computing encounter points for query insertion

A mobile user can issue a new location query Q by sending a message to the server in the form of $(oid, F, range)$. If the location of the mobile client with identifier *oid* in the CT table is older than F, its location information is updated with F and the new query is inserted into *QT* with a new unique query identifier *qid*.

The algorithm to calculate the encounter points and the set of segments covered by the query maintains a queue of $(nid, dist(F, nid))$ tuples, storing node distances from F in ascending order; and a hash-table (initially empty) for segments, where segment identifiers inserted into the hash-table indicate covered segments. The algorithm starts by investigating the distances of the two end-nodes of the segment $n_i n_j$ on which F is located, to detect any encounter points lying on this segment (Figure 2(d)). If $p > range$, then n_i is outside the query range, and an encounter point is at $E = (n_i n_j, p - range)$; otherwise n_i is inserted in the queue. If $length(n_i n_j) - p > range$, then n_j is outside the query range, and an encounter point is at $E = (n_i n_j, p + range)$; otherwise n_j is inserted in the queue.

Tuples are removed from the queue head, and all uncovered segments reachable from the current node n_i are investigated: the segment (of the form $n_i n_j$ or $n_j n_i$) is marked as covered by inserting its sid in the hash-table, and the distance of the segment's other end-node n_j is computed as $dist(F, n_j) = dist(F, n_i) + length(n_i n_j)$. If $dist(F, n_j) > range$, then the segment crosses the query boundary, and an encounter point is located at $E = (n_i n_j, length(n_i n_j) - (dist(F, n_j) - range))$ for $n_i n_j$ (Figure 2(b)), or at $E = (n_j n_i, dist(F, n_j) - range)$ for $n_j n_i$. Otherwise, the segment is entirely covered by the query region, and the tuple $(n_j, dist(F, n_j))$ is inserted into the queue, unless another $(n_j, dist(F, n_j)')$ is already in the queue with $dist(F, n_j) \geq dist(F, n_j)'$. The algorithm terminates when the queue is empty, with the list of encounter points, and the list of (completely or partially) covered segments $\mathcal{E}_q$ stored in the hash-table. Note that the case of two encounter points on a single segment (Figure 2(d)) is handled correctly by adding E_1 when the current node is n_i, and adding E_2 when the current node is n_j.

The segments in $\mathcal{E}_q$ are retrieved from the segment table ST, and the query identifier qid is appended to the list of queries covering the segment.

Using encounter points to answer a query

The set of completely or partially covered segments ($\mathcal{E}_q$) and encounter points of a query are computed using a network expansion algorithm when the server is notified of the query insertion. The initial result of the query is calculated by retrieving all segments of $\mathcal{E}_q$ from ST, then retrieving all oid clients that are listed on these segments. For segments with no encounter points, all mobile clients oid on the segment are added to the result set; otherwise mobile client locations are retrieved from CT to determine if they lie inside the query region. For a client that lies on a segment with a single encounter point, E's location must enclose the client's location, determined by the condition

$$\begin{aligned} &enclosing((E, dir), L) := \\ &\qquad (dir = source \land L_{oid} \leq E.p) \lor \\ &\qquad (dir = target \land L_{oid} \geq E.p), \end{aligned}$$

to be added to the result set. For a client that lies on a segment with two encounter points, we distinguish two cases: if the query coverage extends to an area around the end-nodes (Figure 2(c); $E_1.p < E_2.p \land dir_1 = target \land dir_2 = source$), then one of $enclosing((E_1, dir_1), L_{oid})$ or $enclosing((E_2, dir_2), L_{oid})$ must be true; if the query coverage area is the middle of the segment, with the end-nodes uncovered (Figure 2(d); $E_1.p > E_2.p \land dir_1 = source \land dir_2 = target$), then both $enclosing((E_1, dir_1), L_{oid})$ and $enclosing((E_2, dir_2), L_{oid})$ must be true.

Throughout the iteration over the segments of $\mathcal{E}_q$, a list of precincts that overlaps with the query range is built. The query will be installed on the clients residing within all these covered precincts. However, clients in different precincts will be aware of a different – precinct-specific – set of encounter points associated with this query. Also some covered precincts might be exempt from the need of being query-aware, such as those that do not contain any encounter points. Each precinct is retrieved from PT, and its segments are retrieved from ST. In the first iteration over segments in the precinct, a list of encounter points found in the current precinct are built ($\mathbb{E}_q^{pid}$). If $\mathbb{E}_q^{pid}$ is not empty, then in the second iteration clients on each segment in the current precinct are sent a query-installation message containing $\mathbb{E}_q^{pid}$. Clients residing in precincts that cover the boundaries of the query will be aware of the query. Clients in precincts further away will be unaware of the new query; and if the query range covers a sufficiently large area, some precincts entirely covered by the query (near the central area of the query area) will contain no encounter points, so clients in these precincts will also be unaware.

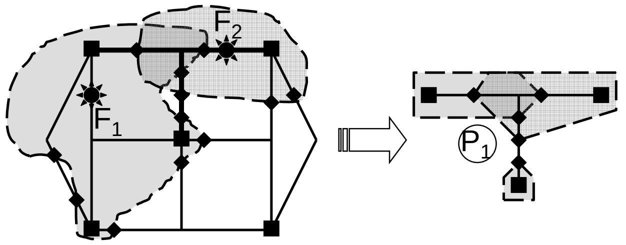


Figure 6: Two overlapping range queries with focal locations F_1 and F_2, and radiuses d_1=1.75 km and d_2=1 km (left), and precinct P_1 with queries displayed (right).

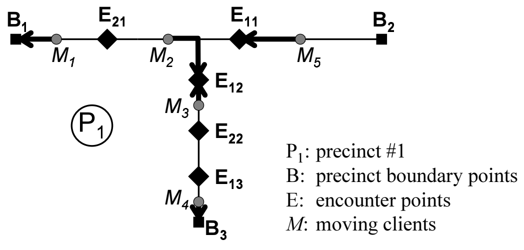


Figure 7: Check-free paths for mobiles $M_1, \ldots, M_5$, that are inside precinct P_1, when queries present are those shown on Figure 6.

4. OPTIMIZATION WITH ENCOUNTER DEPENDENT CHECK-FREE INTERVAL

When a mobile client first becomes a registered user, it submits an orientation request to the server, including her current location. Mobile users registered with the system can be either active or disconnected. A mobile client is required to send a location update message to the server in three cases: (i) When a mobile user is becoming active from a disconnected state, she sends the location database server a location update message of type P. The server responds to a P message by sending the list of encounter points ($\mathbb{E}$) in the user's precinct. (ii) When a mobile user is crossing a precinct boundary ($\mathfrak{U}_B(oid, L)$), she sends the server a location update message of type B. The server responds to the B-message by sending the list of all current encounter points ($\mathbb{E}$) found inside the new precinct. (iii) When a mobile user is crossing an encounter point ($\mathfrak{U}_E(oid, L, E)$), the client sends to the server a location update message of type E. When the server receives an E-message, it updates the result set of the query attached to the E encounter point, either inserting (when entering a query region) or removing (when exiting a query region) oid, and notifying the issuer of the query corresponding to the encounter point of the change in the result set of the query.

A naïve approach to implementing the precinct-based location update scheduling is periodic checking of whether a mobile client has crossed a boundary point or encounter point and thus needs to send a location update to the server. Such decision is typically made based on the motion behavior of the client, the nearby queries and the corresponding encounter points, and the precinct boundary points. An obvious drawback of the periodic checking method is the unnecessary energy and resource consumption at each mobile client, especially when the mobile client is far away from any of the boundary points or encounter points for a given time period. We optimize the periodic checking method by introducing the road network based check-free interval mechanisms, which allows us to significantly enhance the performance of our precinct-based update scheduling algorithm.

Check-Free Road Network Locations

For each mobile user, we can compute a road network based check-free zone, based on its road network location L_c, its movement speed, its trajectory if available, and all the encounter points ($\mathbb{E}$) and boundary points ($\mathbb{B}$) of its current precinct. By check-free, we mean that as long as the mobile client travels within this portion of the network, no location update is necessary. One way to compute the check-free locations of a mobile client is to start from its current network location and perform the following three tasks. First, find the dominating encounter points and boundary points. Second, compute all the paths from the client's current location to every dominating encounter point or boundary point. We call these paths dominating check-free paths. Third, compute the region covered by the dominating check-free paths obtained in the previous step. Intuitively the dominating encounter or boundary points are those that are closer to the current network location of the mobile client. Given two encounter points E_1 and E_2, if the distance of E_1 to L_c is smaller than the distance of E_2 to L_c and the path from L_c to E_1 is covered by the path from L_c to E_2, then we say E_1 is dominating E_2 with respect to L_c.

Check-Free Interval

In order to detect when a mobile user on the move crosses an encounter point or a precinct boundary point, we need to determine when to perform the crossing check. To address the inefficiency of periodic checking for the mobile clients that are far away from any encounter point or boundary point, we introduce the check method based on a *check-free interval* computed for each mobile client. A check-free interval is the longest time that a client can sleep without comparing its location against any dominating boundary or encounter points, while being assured that any such update triggering points are not missed. The check-free interval can be computed as the shortest of the maximum-speed weighted distances (i.e., shortest travel time) to all B and E points within the current precinct. The maximum speed is a road segment specific constant (v_{max}^{seg}) stored with the road network data. The pre-calculated node-to-node distance table D is used for the fast calculation of the *check-free path* lengths (Figure 5). For a given road network location L_c, the check-free interval t_{cf} is computed as follows:

$$t_{cf} = \min_{L \in \mathbb{B} \cup \mathbb{E}} \frac{dist(L_c, L)}{v_{max}^{seg}}.$$

Consider the case of two overlapping queries on the road network with F_1 and F_2 as the focal location respectively as shown in Figure 6. For the purposes of the check-free interval computation, it is actually irrelevant to consider which parts of the segment are inside or outside one, two, or more query regions; only the locations of E and B points are important. The check-free paths for five mobiles (M_1, M_2, M_3, M_4, M_5) in this example are shown as darker line fragments in Figure 7.

Detection of crossing encounter or boundary points

We compute a check-free interval for every mobile client in the context of its current precinct using all the encounter points and boundary points. The mobile client does not need to perform any crossing check with respect to the encounter points and boundary points until its check-free interval is over. The mobile client may enter sleep mode if it does not have other active services. Upon the expiration of its check-free interval, the mobile client needs to determine whether it has crossed any E or B points. If the precinct (pid) of the segment at the last location is different than the precinct of the current location, then the client has crossed at least one B point, and thus a $\mathfrak{U}_B$ update is issued to the server, which in turn sends the encounter point set $\mathbb{E}$ of the new precinct to this client.

If no precinct change has occurred, then we perform the encounter point crossing detection. Given the last and current locations, there may be multiple paths between the two locations and each path may have a different set of E points. Given that the result of a query is independent of which concrete path the mobile has actually taken to move from the last location to the current location, any path between the last and the current location is suitable. We choose the shortest path to collect the E points located on this path. For any set $\mathbb{E}_q$ of encounter points associated with a query, crossing an even number of E points will leave the query result unchanged, since the mobile remains inside (or remains outside) the query range bounded by the two E points both before and after his movement. However, if there are an odd number of E points on this shortest path, this means that the overall movement of the mobile changes the query result. For all queries that have an associated E point on the shortest path, we determine the number of E points owned by that query. If any of these numbers is odd, then a query result has changed, and an $\mathfrak{U}_E$ update is issued.

5. EXPERIMENTAL EVALUATION

In this section we present the experimental evaluation of our query-aware location update approach through four sets of experiments. We first compare our ROADTRACK location update approach with the four representative update strategies discussed in Section 2 in terms of number of updates per unit time at both server and client under two types of road networks: urban and rural. We show that the query-aware location update strategy significantly outperforms existing update strategies in terms of both client computation cost (#wakeups) and server updates for both urban and rural road networks. The second set of experiments measures the scalability of ROADTRACK by varying the number of mobile objects in the system. The third set of experiments examines the effect of different mobility models of mobile clients, different query characteristics, and the precinct size on the effectiveness of our query aware location update approach. Our experiments show that the query-aware update strategy offers consistent performance in terms of both server update load and client wakeup load under different road network mobility models, different precinct sizes , different query loads, different query radius, and different query distribution models (uniform and hotspot). The last set of experiments examines the cost of precinct construction in terms of computation time, average number of nodes, number of precincts, size of precinct.

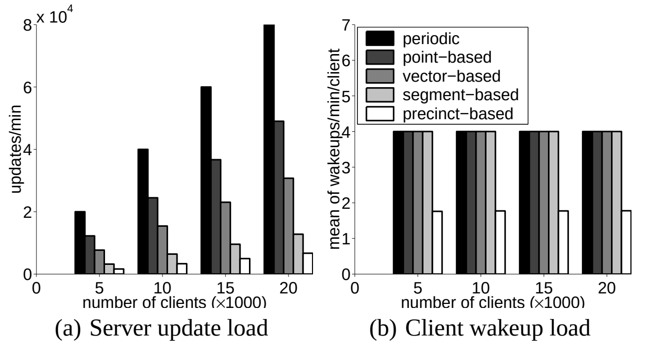
5.1 Experiment setup

We use real road networks obtained from the US Census Bureau's TIGER/Line collection [7] in our experiments (Table 1). Maximum speeds are specified for each of four road classes at 30 mph for residential, 55 mph for highway, 70 mph for freeway, and 30 mph for freeway interchange (i.e. 48, 89, 113 and 48 km/h).

We created an event-based simulator for the evaluation of our framework. Instead of applying a timestepping approach, a central ordered event queue is used to schedule four types of events: change in the mobility pattern of an object (velocity vector change), query insertion, query deletion, and client wakeup. The single-threaded simulation consists of removing the events from the queue head, taking the assigned event action (eg. run client code on client wakeup, which might issue an update, which in turn causes the execution of server code), and inserting new events into the queue (such as the next requested wakeup with a future timestamp). The queue is initiated with the mobility pattern change and query insertion/deletion events, generated by a *mobility model* and *query model*, for the entire requested duration of the simulation. We consider two mobility models in this paper: random waypoint move-

Table 1: Road networks used in experiments

Style	County location	Total length	Segments	Junctions	Avg. segment length	Junction degree
urban	Miami-Dade, FL	15 650 km (315 h)	109 416	79 101	143.0 m (10.4 sec)	mean: 3.4, max: 8
rural	Coconino, AZ	36 212 km (733 h)	81 918	67 911	442.1 m (32.2 sec)	mean: 2.4, max: 6



(a) Server update load (b) Client wakeup load

Figure 8: Scaling of update strategies with number of mobile clients (rural map, partition $r = 4\times$)

ment on road network (RWR), and random trip model on road network (RTR). In both mobility models, each mobile object moves independently of others, with a speed that changes only when entering a new segment, and which is chosen according to the speed limit and speed distribution defined for the segment. In a RWR model, the client selects and travels a new segment at random at each junction; then repeats. In a RTR model, the client chooses a random trip destination on the map, travels the fastest route; then repeats. Client speeds are chosen from a bell-curve distribution (a Gaussian with a standard deviation of 0.2 times the mean) that is cropped above its mean (segment speed limit).

The query model we used maintains a 10% location query load in the system by default (i.e., the number of queries is one tenth of the number of mobile clients). Note that this is an aggressive query load, as it signifies that our system actively engages the attention of $\frac{1}{10}th$ of the population at any one time. Query ranges are chosen from a Gaussian distribution with a mean of 1 km, and standard deviation of 0.1 km. In order to simulate a more realistic scenario than that given by a uniform distribution of query centers, we create a query hotspot scenario, whereby queries are highly concentrated in some region of the map. The center of a hotspot is a road network location chosen from a random distribution over all road network locations in the network. Once the hotspot center is established, a weight is assigned to each road segment in the network. If the shortest road network distance between the hotspot center and the mid-point of a segment is d in kilometers, then we assign the weight $w_i = \alpha^d$, with $\alpha = 0.5$. Each segment then has a $\frac{w_i}{\sum_i w_i}$ chance of being selected as a query center location. Finally, the mobile object closest to the midpoint of the selected segment is chosen as the query's originator.

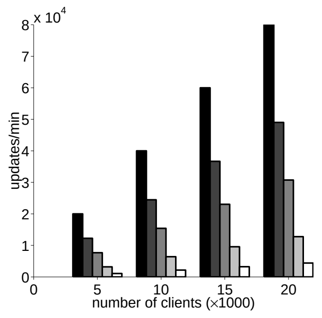
5.2 Messaging cost of update strategies

We compare the number of client wakeups and server update loads for various location update approaches by varying the number of users in the system. This set of experiments uses a client population with size ranging from 5000 to $20\,000$ clients. We compare the following strategies: (1) periodic updates every 15 seconds; (2) point-based tracking, (3) vector-based tracking, (4) segment-based tracking, and (5) encounter point-based wakeup and update strategy. For the first four query unaware approaches, the wakeup frequency and the reevaluation frequency at the server is set at 15 seconds, and the deviation threshold is set to 25 m. For the query-aware RoadTrack approach, we set a maximum wakeup frequency of once every 15 seconds (4/min), in order to allow performance of all methods at similar operating points with regards to accuracy. The comparison on the rural Coconino County map, with partition radii of 4 times the mean segment length (i.e., $r = 1768\ m$) shows that the encounter based method results in a significantly reduced rate of wakeups (Figure 8(b)). The advantage of RoadTrack is the that wakeups are unnecessary when a client is distant from encounter points (query boundaries) and precinct border nodes. Note that periodic, point-, vector-, and segment-based approaches all produce 4 wakeups per minute due to their 15 sec reevaluation setting – since a check-free interval type optimization is not available, they wastefully execute periodic self-checks.

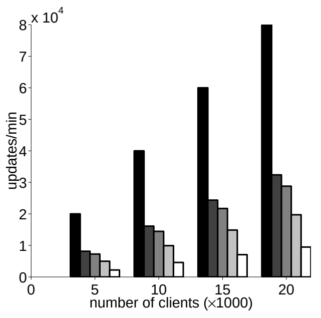
The number of server side updates is shown in Figure 8(a). This experiment confirms the conceptual insight that the precinct-based RoadTrack approach outperforms all existing approaches even in the worst case. Note that since the query load is a constant 10%, the increase in the number of mobile clients also brings a proportional increase in the number of queries at the same time. As a result, not only does the number of mobiles per precinct increase, but the number of encounter points per precinct also increases. As each encounter point is an update trigger, the number of updates issued also necessarily increases. The RoadTrack strategy allows a reduction to 8%, 14%, 22%, and 52% of updates, relative to the other four comparison methods, respectively, even at the highest mobile load studied.

We further explore the scalability of our system by using precincts with radii that are 8 times the mean segment length (i.e., $r = 3536\ m$), and also running the simulations on the urban Miami-Dade County map (where $r = 1144\ m$). The larger precinct size provides a significant boost for RoadTrack: Wakeups are reduced on longer check-free intervals, as border points are – on average – further from mobile clients (Figure 9(c)). At the highest load setting, updates are reduced to 6%, 9%, 14%, and 34% of the query unaware approaches' updates (Figure 9(a)). The $\frac{1}{3}$ mean segment length of the urban, high density topology means that the distance-based partitioning creates segments that cover smaller areas, and thus the average distance from mobile clients to precinct boundaries increases. The high density also means that a query with the same radius (as measured on the roads) produces more encounter points. These factors bring an increase in the number of both wakeups and updates when compared with the low-density rural map, but the update count is still significantly lower in comparison with the four reference strategies (11%, 28%, 31%, and 46% of the updates of those methods, Figure 9(b) and 9(d)).

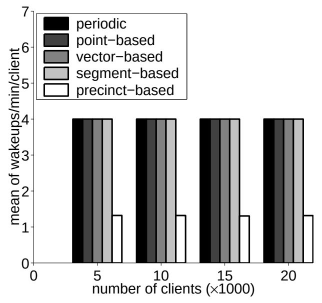
We also plot the effect of precinct seeding and the partitioning metric, when no queries are present, and thus all updates and wakeups are triggered by precinct boundaries only (Figure 10). We point out that the presence of precinct boundaries causes wakeups and updates even without the presence of queries. This property, by design, ensures that the server maintains the location tracking ability for all mobile clients, regardless of whether there are queries nearby or not. The benefit of an encounter-based strategy over strategies



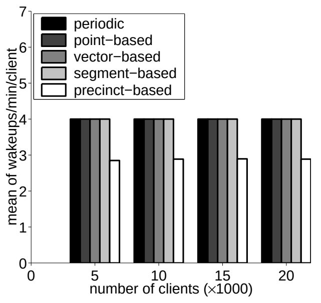
(a) Server updates (rural)



(b) Server updates (urban)

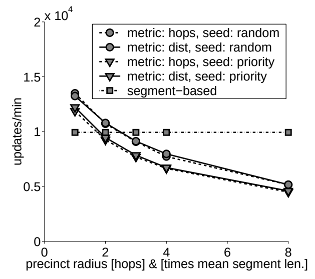


(c) Client computation cost (#wakeups, rural)

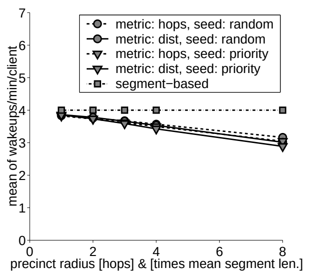


(d) Client computation cost (#wakeups, urban)

Figure 9: Comparison of rural and urban performance (with 8× distance-metric partitioning for the two maps in Table 1)



(a) Server updates



(b) Client wakeups

Figure 10: Comparison of precinct-triggered and segment-based strategies (urban map, precinct $r = 8\times$)

that are query-unaware (such as periodic, point-based, etc. methods) is the reduction of unnecessary wakeups and updates. On the other hand, if updates were only issued at query boundaries, in the case of very few queries in a region, a client could go for an extended period of time without an update, and the server would be unable to contact the client for a location update in order to answer a new query. The requirement to issue updates at precinct boundaries not only allows a client to be aware of query border points in its vicinity (after the server sends this information about the new precinct), but also allows the server to maintain an approximate location (bounded by the current precinct's boundary) of the client's whereabouts. These figures consistently show that larger partitions help reduce both updates and wakeups. We compare our precinct-based strategy with segment-based updates – at a radius of 1 hop, precinct-based updating is very similar to segment-based updates. As a result, the number of wakeups are only slightly improved from segment-based periodic wakeups when the precinct radius is small, but the improvement gap increases linearly with precinct size (Figure 10(b)). The number of updates is higher for low precinct sizes, as the $delta$ threshold used for segment-based updates (and the resulting inaccuracy) is not present in RoadTrack, but the update count drops to half of the segment-based updates at a radius of 8 times mean segment length (Figure 10(a)).

In the following we concentrate on the urban map, which is a more challenging terrain for RoadTrack due to the higher density network topology.

5.3 Effect of client behavior profile

We investigate our method with respect to its sensitivity to different characteristics of client behavior in two dimensions: mobility model and query radius distribution (Figure 11(c) and 12(c)). For mobility models, we consider the RWR and RTR type behaviors; for query size distributions, we vary the mean of the Gaussian distribution defining query radii chosen by clients, while keeping the standard deviation of the Gaussian at 10% of the mean. Our qualitative conclusion is that the RWR movement model is slightly more advantageous for our approach, but this advantage decreases as query radii increase at client side. In all other experiments reported in this paper, we employ the RTR movement model, to avoid any unfair comparisons.

5.4 Effect of precinct size and query load

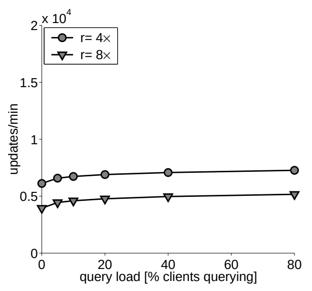
We investigate the effect of precinct size on our metrics with $10\,000$ clients, uniform and hotspot query distribution (Figure 11(b) and 12(b)). The simulations show that a hotspot distribution of query centers takes advantage of the features of our approach, producing fewer updates and wakeups.

We inject a query load varying from 0% to 40% (i.e., 0 to 4000 queries), and run measurements using distance metric partitioning with the radius set to 4 and 8 times the mean segment length (i.e., $r = 572\ m$ for $4\times$, and $r = 1144\ m$ for $8\times$), with the results shown in Figure 11(a) and 12(a). The number of wakeups decreases with growing precinct size, as the influence of precinct boundaries on the check-free interval decreases. As many wakeups are false alarms (an update is not actually required), the number of wakeups is less impacted by an increase in query load, than the number of updates (Figure 11(a)).

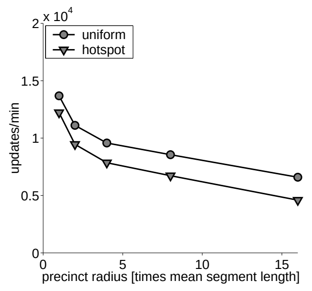
5.5 Precinct construction

The number of partitions created as a function of the requested precinct radius is shown in Figure 13(a). Distance-based partitioning is shown as a function of distance values that are multiples of the average segment length of 143 m, offering convenient comparison with hop-based partitioning, shown as a function of the hop count (e.g., partitioning with "3 [hops]", and "3 [times mean segment length] ($= 3 \cdot 143\ m$)" settings are shown at the same X axis value). The average number of graph nodes per precinct grows only linearly with the precinct radius, largely due to the skew effect of more "leftover" smaller precincts when the requested precinct size is large (Figure 13(c)). The storage space required for the precomputed node-to-node distance matrices is defined by the number of node pairs per precinct. This storage requirement (Figure 13(d)), and the wall clock time required to compute it (Figure 13(b)) grow approximately as the square of precinct size. We remark that when precinct center nodes are selected using our heuristic-based seeding method, the number of precincts is reduced. In our experiments – unless noted otherwise – we thus used heuristic-based precinct seeding, and distance-metric partitioning with 8 times the mean segment length (i.e., $r = 1144\ m$).

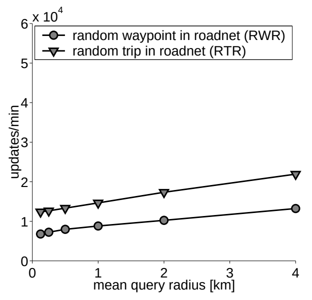
We also provide a comparison with partitioning the large rural



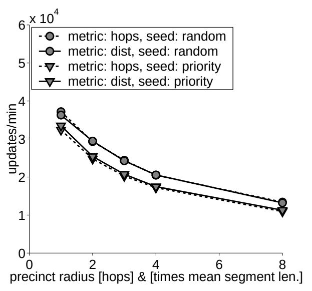
(a) Scaling with query load and precinct radius



(b) Effect of precinct size and query distribution

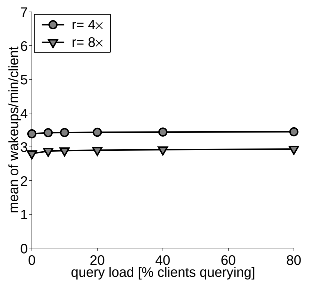


(c) Effect of client behavior characteristics

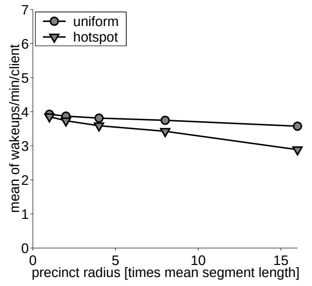


(d) Precinct-triggered updates

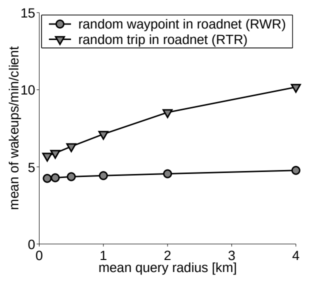
Figure 11: Server update load [fastest wakeup setting: $(a), (b)$: 15 sec, $(c), (d)$: 5 sec]



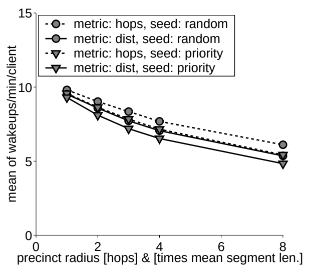
(a) Scaling with query load and precinct radius



(b) Effect of precinct size and query distribution



(c) Effect of client behavior characteristics



(d) Precinct-triggered wakeups

Figure 12: Client wakeup load (urban map) [fastest wakeup setting: $(a), (b)$: 15 sec, $(c), (d)$: 5 sec]

map (Figure 13(e)–13(h)), and note that while the average number of nodes per precinct is almost independent of the partitioning method, the number of node pairs (and the resulting increased pre-computation time) increases markedly with distance-based partitioning for our rural map, due to the different topology.

6. RELATED WORK

We review three threads of related work, which are most relevant to the location update efficiency.

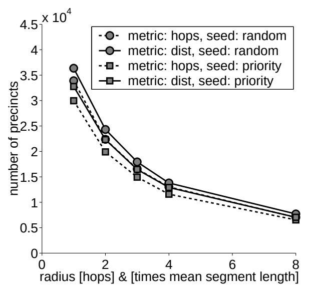
The first group of work explores the idea of reducing the number of location updates in presence of a road network, but without making mobile clients query aware. [2] gives a good summary of the techniques. As we show in this paper, the number of location updates can be significantly reduced when clients are query-aware, as there is no need for clients to issue updates in locales where outstanding queries are scarce. Even in the worst case, the precinct based approach outperforms the existing solutions.

The second group of work explores the idea of reducing query processing load at the server by making clients query-aware but in a world where constraints on client mobility do not exist (i.e., without a road network). For example, MobiEyes [8] uses the grid structure to define a monitoring region for each query, and only clients within the monitoring region need to be aware of the query. [9, 10, 11] give a solution for static continuous queries over moving objects, by monitoring violations of safe region areas. As we show in this paper, when road constraints on queries exist (such as a distance or travel time range measured in the network), the solution must address the jump in complexity: we use encounter points to implement the query awareness and identify the critical points on road network segments where location update should be performed. In addition, we use precincts to impose locality on query-awareness and to set the upper bound for mobile clients to update their locations.

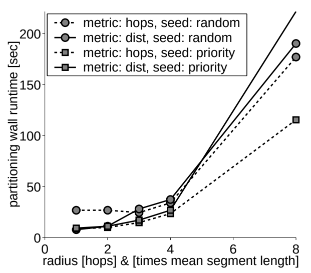
The third group of work explores the idea of reducing server load for query processing in the presence of a road network. The incorporation of road networks in server optimization of mobile queries started to gain attraction only in recent years [12, 13]. A most influential line of work in this group is the idea of speeding query answering by pre-computing distances after partitioning the road network graph [13]. However, no consideration is given to improve the server load for query processing by utilizing a road network based, query-aware location update scheme. We believe that the ROADTRACK development can be beneficial for further reduction of server load for processing location queries on road networks.

7. CONCLUSION

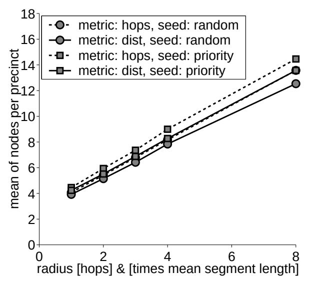
In recent years, some LBS providers in European countries have initiated a pay-as-you-go model for location tracking and location update services, with the primary objective of avoiding unexpected sudden load surges at location servers. For example, mobile users can pay a fixed price for being tracked or for keeping their location updated every five or 10 minutes. With the rapid escalation of location based applications and services and the growing demand of being informed at all times, the problem of scaling location updates and location tracking systems and services, if not addressed, will become a performance bottleneck for the success of the mobile commerce and mobile service industry. In this paper we have presented ROADTRACK – a query-aware, precinct based location update framework for scaling location updates and location tracking services. ROADTRACK development makes three original contributions. First, we introduce encounter points as a funda-



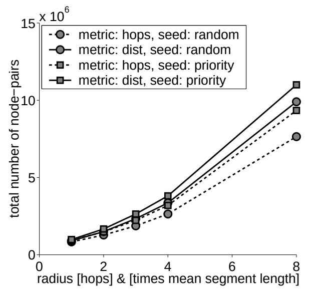
(a) Number of precincts



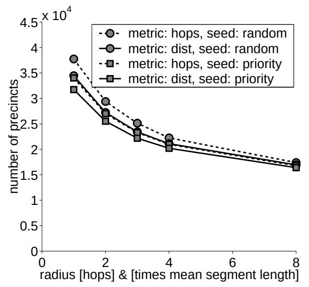
(b) Partitioning and pre-computation wall clock time



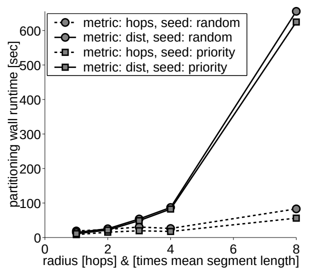
(c) Average number of nodes per precinct



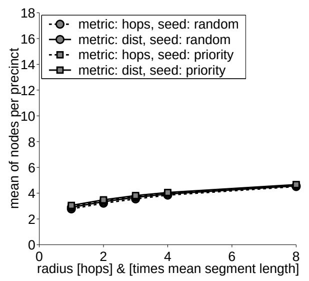
(d) Total number of node pairs



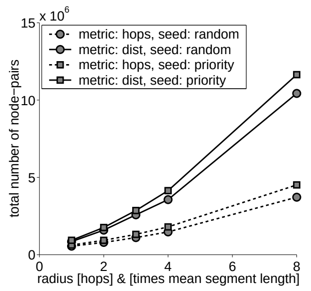
(e) Number of precincts



(f) Partitioning and pre-computation wall clock time



(g) Average number of nodes per precinct



(h) Total number of node pairs

Figure 13: Effects of precinct radius on partitioning with hop and distance metrics ($(a)-(d): urban, (e)-(h): rural$**)**

mental query awareness mechanism enable us to control and differentiate location update strategies for mobile clients in the vicinity of active location queries. Second, we employ system-defined precincts to manage the desired spatial resolution of location updates for all mobile clients and to control the scope of query awareness capitalized by a location update strategy. Third but not the least, we develop a road network distance based check-free interval optimization, which further enhances the effectiveness of ROADTRACK and enables us to effectively manage location updates of mobile clients traveling on road networks by minimizing the unnecessary checks of whether they have crossed an encounter point or precinct boundary point. We evaluate the ROADTRACK location update approach using a real world road-network based mobility simulator. Our experimental results show that the ROADTRACK query aware, precinct-based location update strategy outperforms existing representative location update strategies in terms of both client computation efficiency and server update load.

Acknowledgement

The authors from Georgia Institute of Technology acknowledge the partial support by grants from NSF NetSE, NSF CyberTrust, an IBM SUR grant, an IBM faculty award, and a grant from Intel Research Council.

8. REFERENCES

[1] A. Bar-Noy and I. Kessler, "Mobile users: To update or not to update?" 1994.

[2] A. Civilis, C. S. Jensen, and S. Pakalnis, "Techniques for efficient road-network-based tracking of moving objects," *Proc. IEEE TKDE*, vol. 17, no. 5, pp. 698–712, 2005.

[3] Skyhook Wireless, "Hybrid Positoning System (XPS)," http://www.skyhookwireless.com/howitworks/.

[4] K. Jones and L. Liu, "What Where Wi: An analysis of millions of wi-fi access points," in *IEEE PORTABLE*, 2007.

[5] P. Pesti, B. Bamba, M. Doo, L. Liu, B. Palanisamy and M. Weber, "GTMobiSIM: A mobile trace generator for road networks," College of Computing, Georgia Inst. of Tech., Sept. 2009, http://code.google.com/p/gt-mobisim/.

[6] A. Murugappan and L. Liu, "An energy efficient approach to processing spatial alarms on mobile clients," in *Proc. of ISCA International Conference on Software Engineering and Data Engineering*, 2008.

[7] U.S. Census Bureau, "TIGER/Line Shapefiles," http://www.census.gov/geo/www/tiger/.

[8] B. Gedik and L. Liu, "Mobieyes: A distributed location monitoring service using moving location queries," *Proc. IEEE Trans. Mobile Computing*, vol. 5, no. 10, pp. 1384–1402, 2006.

[9] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *SIGMOD*, 2005.

[10] Y. Cai, K. A. Hua, G. Cao, and T. Xu, "Real-time processing of range-monitoring queries in heterogeneous mobile databases," *Proc. IEEE Trans. Mob. Comput*, vol. 5, no. 7, pp. 931–942, 2006.

[11] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch, "Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects," *Proc. IEEE Trans. Computers*, vol. 51, no. 10, pp. 1124–1140, 2002.

[12] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB*, 2003.

[13] M. R. Kolahdouzan and C. Shahabi, "Voronoi-based K nearest neighbor search for spatial network databases," in *VLDB*, 2004.