A Energy Efficient Approach to Processing Spatial Alarms on Mobile Clients

Anand Murugappan and Ling Liu
College of Computing, Georgia Institute of Technology, Atlanta, USA
E-mail: {anandm, lingliu}@cc.gatech.edu

Abstract

Many on a daily basis use time based alarms. Spatial alarms extend the very same idea to location-based triggers, which are fired whenever a mobile user enters the spatial region of the location alarms. Spatial alarms provide critical capabilities for many mobile location based applications ranging from personal assistants, inventory tracking to industrial safety warning systems. In this paper we present an energy efficient framework for processing spatial alarms on mobile clients, while maintaining low computation and storage costs. Our approach to spatial alarms provides two systematic methods for minimizing energy consumption on mobile clients. First, we introduce the concept of safe distance to reduce the number of unnecessary mobile client wakeups for spatial alarm evaluation. This mechanism not only reduces the amount of unnecessary processing of the spatial alarms but also significantly minimizes the energy consumption on mobile clients, compared to periodic wakeups, while preserving the accuracy and timeliness of the spatial alarms. Second, we develop a suite of techniques for minimizing the number of location triggers to be checked for spatial alarm evaluation upon each wakeup. This further reduces the computation cost and energy expenditure on mobile clients. We evaluate the scalability and energy-efficiency of our approach using a road network simulator. Our client based framework for spatial alarms offers significant improvements on both system performance and battery lifetime of mobile clients, while maintaining high quality of spatial alarm services, especially compared to the conventional approach of periodic wakeup and checking all alarms upon wakeup.

1 Introduction

Spatial alarms are considered by many as one of the critical mobile location-based applications in future computing environments. Processing of spatial alarms requires meeting two demanding objectives: high accuracy, which ensures no alarms are missed and high energy efficiency and high scalability, which not only minimizes the unnecessary processing cost and the consumption of energy on spatial alarm processing but also scales the energy efficient processing to larger number of spatial alarms on mobile clients. The conventional approach to designing the middleware architecture for spatial alarms involves periodic alarm checks at a high frequency. Each spatial alarm check is conducted by testing whether the user is entering the spatial region of the alarm. High frequency is essential to ensure that none of the alarms are missed. This technique is simple but can be extremely energy inefficient due to both frequent wakeups and evaluation of all alarms upon each wakeup. This is especially true when the mobile client is traveling in a location that is distant from the spatial areas of all her location triggers or when the collection of spatial alarms is set on spatial regions that are far apart from one another. In addition, some types of mobile clients have stronger resource constraints such as smart phones and hand held PDAs compared to navigation systems in cars.

In this paper we present our architecture for energy efficient processing of spatial alarms on mobile clients, while maintaining low computation and storage costs. We present two systematic methods that can progressively minimize the amount of energy consumption on mobile clients for all types of spatial alarms. The first method utilizes the concept of safe distance to reduce the number of unnecessary wakeups on mobile clients for spatial alarm evaluation. By enabling mobile clients to sleep for longer intervals of time in the presence of active spatial alarms, we show that our safe distance techniques can significantly minimize the energy consumption on mobile clients compared to periodic wakeups, while preserving the accuracy and timeliness of spatial alarms. The second mechanism focuses on alarm checks upon each wakeup. We develop a suite of techniques for minimizing the number of location triggers to be checked upon each

wakeup for different types of spatial alarms. This allows us to further reduce the computation cost and energy expenditure on mobile clients. Our experimental evaluation using a road network simulator shows that our spatial alarms middleware architecture offers significant improvements on battery lifetime of mobile clients, while maintaining high quality of spatial alarm services compared to the conventional approach of periodic wakeup and checking all alarms upon a wakeup.

2 System Model

A spatial alarm consists of three components: the spatial region on a two-dimensional geographical plane, the action to be taken upon firing of the alarm, and the alarm termination condition, usually a temporal event such as time point or time interval. The spatial regions used in spatial alarms can be of any shape. We capture each of such spatial regions by a rectangular bounding box, denoted by (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) represent the top-left and bottom-right vertices of the bounding rectangle. Without loss of generality, in the rest of the paper, we simply assume that each mobile client can install n spatial alarms ($n \geq 0$) and all spatial alarms are expressed by a rectangle spatial region, denoted by A_i for $1 \leq i \leq n$. In our first spatial alarm client middleware prototype, we use a system supplied default spatial range in the absence of spatial region specification of a user-defined alarm. Each mobile client can install as many spatial alarms as the user wishes over the geographical area of interest. Multiple mobile clients can set spatial alarms on the same locations.

In this paper we assume that mobile clients have limited energy, storage, and computational resources. We also assume that at least one of positioning technologies, such as GPS, WiFi based triangulation or cellular network assisted location identification services, is available for the mobile client to identify its current location. Each mobile client is a moving object with an accompanying mobile device, such as cell phone, PDA, car, which computes and communicates the current location of the mobile client with the Spatial Alarm client middleware. A mobile client can move freely in the entire geographical area of interest (universe of discourse). In addition, mobile clients may have a map service available on their portable device, through which the mobile client can navigate on the map to identify and install their personalized spatial alarms through interactive mode of communication with the Spatial Alarm client middleware. Batch installation of spatial alarms is also possible. Figure 1(a) shows the spatial alarms installed in the Georgia Tech area on the real world maps obtained from Google Maps [1]. The rectangles denote the user specified alarm areas. Figure 1(b) shows the corresponding road network map from the TIGER database [2], which our simulator uses for evaluating the effectiveness of the proposed architecture and algorithms.

Although our development of the Spatial Alarm framework considers three alternative architectures: client-centric, server-centric with thin client, and fat client with server mediation. In this paper we concentrate on the client-centric architecture in the sense that the server is not involved in spatial alarm processing at all, though we allow the mobile clients to utilize some third party auxiliary services, such as the map service, the Voronoi diagram provision service, to name a few. We also assume that the client has the spatial alarm storage manager that provides persistent store for all its spatial alarms. On demand or periodic offload of client alarms on the server is provided to allow mobile clients to off load their alarms to a server in the event of client storage shortage.

3 System Overview

Spatial alarms differ from spatial location queries in a number of ways. First, spatial queries such as "tell me the gas stations within 10 miles on the highway 85 north" require continuous evaluation of the queries as the mobile client moves on the highway 85 north. However, spatial alarms, such as "notify me whenever I am 5 miles away from this particular dry cleaning store (marked on the map)", only requires the alarm to be evaluated when the mobile client moves to a region that is within 5 miles of the specific dry cleaning store. Thus, it is no use to wake up a mobile client if she is 30 miles away from the dry cleaning store. Clearly, the movement patterns of the mobile client and the distance from the current location of a mobile client to all her alarms are the two critical factors that will affect when the mobile client needs to wakeup and what alarms need to be checked upon each wakeup. Thus one can optimize the spatial alarm processing by devising more energy efficient algorithms.

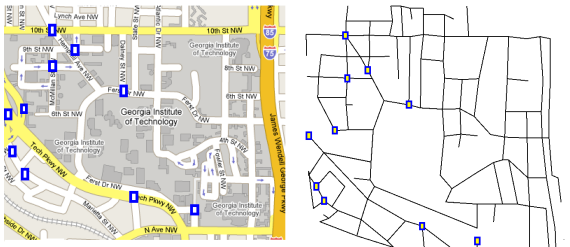
Mobile devices conserve energy by spending most of their time in a low energy state such as sleep mode. Hence one of the critical design objectives for client middleware architecture is to minimize the number of device wakeups in spatial alarm processing. For instance, the 206 MHz Itsy [4] pocket computer spends 540mW power in the *System Idle, 0% processor idle* state, spends 100mW power in the *System Idle, 95% processor idle* state, and while in the *Sleep mode* it just spends 8.39mW power (which is about 64 times lesser than the 0% processor idle case). It is interesting to note that mobile devices like in the case of Itsy computer [4] have a battery lifetime of only 3.8 hrs when running in the high energy *System Idle, 95% processor*

idle state, while in the *Sleep mode* the lifetime is as high as 279 hrs.

In the context of spatial alarm processing, one can save energy by two phase optimizations. In Phase one we minimize the number of device wakeups and in phase two we further minimize the number of alarms checked upon each wakeup. The phase one optimization helps keep the mobile client in the 'Sleep mode' as long as possible, while the phase two optimization helps reduce the computation cycles used for alarm evaluation, which further reduces the energy requirements. In addition to energy efficiency, another important goal of spatial alarm processing is to maintain the low or zero alarm misses.

The conventional approach for implementing a location based reminder system is to wake up the device and check the alarm conditions periodically. If the period is too large the mobile device might miss alarms since there may be situations where the mobile client passes through the 'alarm area' while asleep (between periodic checks). Hence, to reduce the number of alarm misses, the wakeup period would have to be kept small enough. The smallest wakeup period can be set using the location update frequency (e.g., GPS sampling period). Clearly, the periodic check approach would be very energy inefficient. Also it is important to note that if the mobile client is far away from any of her alarms then depending on the maximum speed of the client, it is possible to sleep for longer durations of time and still guarantee that none of the alarms would be missed.

Motivated by this observation, we propose the concept of safe period which computes the time period during which the mobile client can continue to sleep without missing any of her alarms. We compute this safe period using the distance from the current location of the mobile client to all her alarms and the speed measure of the mobile client. In our prototype we consider Euclidean distance and road network distance as two alternative distance functions and consider maximum travel speed and expected travel speed of a mobile client as two alternative speed functions. The expected speed measure is used to handle the situation where the mobile clients do not travel at their maximum possible speed at all times, hence by considering the 'Average Speed' or the 'Expected Speed', we may present a wakeup algorithm that is more adaptive to the movement behavior and the actual distance from the client to the alarms. Further, by considering road network distance as a more accurate prediction of the distance from the mobile client to all her alarms, we can further extend the sleep time without any alarm misses. We discuss these alternative approaches to minimizing alarm wakeups in greater detail in section 4.



(a)From Google Maps[1]

(b)From TIGER [2]

Figure 1: Spatial Alarms

In addition to minimizing wakeups in the phase one optimization, we have noted that it is also essential to reduce the computations performed for processing alarms at each wakeup. The naive approach requires checking against *all* alarms upon each wakeup, which can be expensive. One way to mitigate this problem is to group the alarms based on their spatial proximity, check them together in groups and drill down until individual alarm checks are performed. We refer to the step of minimizing alarm checks as the 'phase two optimization'. We extend three popular indexing algorithms to perform the *alarm grouping optimizations* using R-Tree [14], Voronoi diagram that partitions the two-dimensional coordinate space into Voronoi Regions [3, 12], resulting in a very efficient O(1) nearest alarm lookup. To mitigate the additional storage cost, we use the road network information to generate a Network Voronoi Diagram [9], which achieves very similar performance numbers as the Euclidean distance metrics, but with reduced storage costs. We discuss these optimizations in greater detail in section 5.

4 Minimizing Device Wakeups

Our architecture for spatial alarm processing consists of two phase optimizations. In this section we discuss the phase one optimization strategies that minimize the number of device wakeups. Apart from the high energy consumption, an important problem with the periodic wakeup approach is that it is hard to estimate how frequently the device should wakeup to ensure no alarms will be missed. Two factors that are critical in determining such a frequency: (a) The speed of the mobile client; and (b) the size of the spatial alarm region. Unless the frequency is set to be extremely high (close to the location update frequency), it would always be possible to introduce cases where alarms can be missed by having alarms of the size smaller than the distance traveled by the mobile client between two consecutive wakeups. Thus the key challenge is to determine the right time for mobile clients to wakeup in terms of energy efficiency and alarm ac-

curacy and given a location update, how to determine the subset of alarms that should be checked to conserve energy while maintaining zero alarm misses.

With both the problem of guaranteed alarm delivery and that of energy conservation in mind, we propose four optimization strategies to estimate the safe period and use aperiodic wakeup of the mobile client based on (a) the distance of the client to all her alarms and (b) the travel speed of the mobile client.

4.1 Measuring the Distance to Alarm

There are two most commonly used methods for measuring the distance from a mobile client's current location to an alarm. They are (a) Euclidean Distance and (b) Road Network Distance. The *Euclidean Distance* approach is simpler and requires much lesser data but may at times underestimate the time to sleep before the next wakeup. The *Road Network Distance* measure offers a more accurate estimate of the distance from a mobile client's current location to the spatial region of the alarm, but it introduces additional overhead with handling the road network map data. We propose techniques to mitigate this additional overhead by dividing the original map into tiles and selectively download relevant tiles to a mobile client.

4.1.1 Euclidean Distance to an Alarm

Given a spatial alarm A_i with rectangular spatial alarm region represented by four vertices of the rectangle: (P_1, P_2, P_3, P_4) where $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_1)$, $P_3 = (x_2, y_2)$ and $P_4 = (x_1, y_2)$. Let the mobile client be at P_m represented by the coordinates (x_m, y_m), then the Euclidean distance from P_m to the alarm region of A_i, denoted by d_{A_i}, can be computed by considering four cases. *Case 1:* when the mobile device is within the alarm boundaries the distance to the alarm is zero; *Case 2:* when the mobile device is within the y scope (represented using dotted lines in Figure 2 a)) the distance is the shortest of the distances to alarm edges parallel to the x axis from the mobile client; *Case 3:* when the mobile device is within the x scope the distance is the shortest of the distances to alarm edges parallel to the y axis from the mobile client; and *Case 4:* when the mobile device is outside both the x and y scopes, then the distance is the minimum of the Euclidean distances to the four vertices. The four cases can be formally defined as follows:

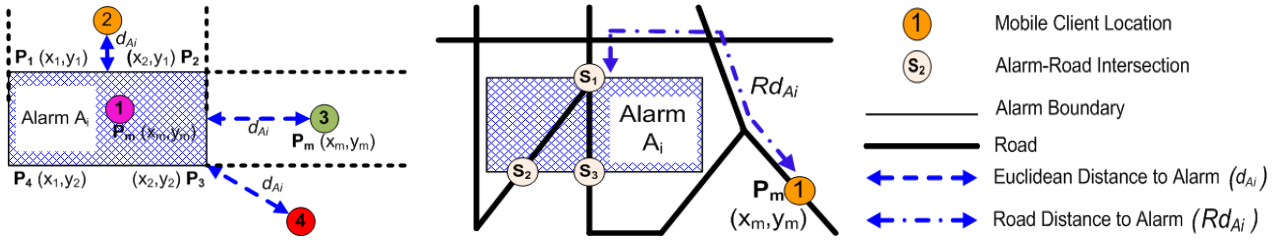
$$d_{A_i} = \begin{cases} 0 & x_1 \leq x_m \leq x_2 \\ & \textit{and } y_1 \leq y_m \leq y_2 \\ min(|x_m - x_1|, |x_m - x_2|) & y_1 \leq y_m \leq y_2 \textit{ only} \\ min(|y_m - y_1|, |y_m - y_2|) & x_1 \leq x_m \leq x_2 \textit{ only} \\ min(D_{m1}, D_{m2}, D_{m3}, D_{m4}) & otherwise \end{cases}$$

Where $D_{m1}, D_{m2}, D_{m3}, D_{m4}$ denote the Euclidean distance from P_m to the four rectangle vertices P_1, P_2, P_3, P_4 respectively. The distance function $D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ is used to compute the Euclidean distance between two points P_i and P_j.

4.1.2 Road Network Distance

One of the main weaknesses of the *Euclidean Distance* measure is that the estimated distance is often shorter than the actual distance that the mobile client would have to travel to get to the spatial region of interest of a given alarm due to the underlying traversal restrictions imposed by the road network. The *Road Network Distance* measure uses the Dijkstra's shortest path algorithm [8, 6] to estimate the distance from the mobile client's current location to an alarm as shown in Figure 2 (b). The underlying road network is represented by the solid line and the mobile client is represented by a shaded circle labeled by 1. Since the mobile client is restricted to move along the roads, the only places where it can enter the alarm area would be the points of intersection of the alarm with the roads, denoted by S_1, S_2, S_3 in Figure 2 (b). Hence, in order to estimate the *Road Network Distance* (Rd_{A_i}) from Mobile Client at P_m to an alarm A_i we calculate the shortest network distance (using Dijkstra's algorithm) to each of the three points of intersection and choose the minimum. Mathematically, $Rd_{A_i} = min(SPath(P_m, S_1), \ldots, SPath(P_m, S_j), \ldots, SPath(P_m, S_k))$ where k is the total number of intersections of the alarm area with different roads on the map, S_j represents the j^{th} point of intersection ($1 \leq j \leq k$), and $SPath(P_m, S_j)$ representing the *Shortest Road Distance* from Mobile Client's location at Point P_m to the j^{th} alarm-road intersection point S_j obtained using the Dijkstra's Shortest path Algorithm [6, 8].

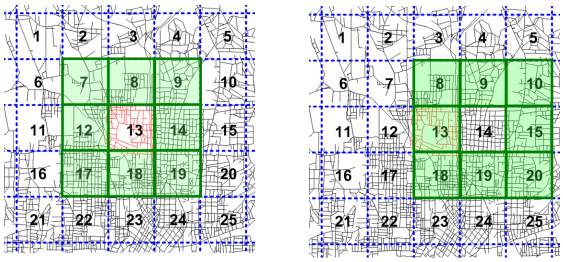
Since computing the road network distance requires detailed maps to perform the calculations, we need efficient mechanisms to handle the situation when it is not possible to store the entire available map information on the mobile client. One approach to mitigate this problem is to divide the entire map into square map tiles and fetch only the relevant tiles to the memory of the mobile client each time when the road network distance is computed. If the mobile client does not have memory to host the map for the entire geographical area of interest, a third party map service can be used for mobile clients to fetch the relevant titles from the server. Consider the example shown in the Figure 3. Consider the midtown Atlanta map as the area of interest. To begin with, the mobile client requests the server to send over nine equal sized square tiles that form a 3 by 3 tile matrix on the map with the tile in which the mobile client resides as the inner most



(a) Euclidean Alarm Distance (b) Road Network Alarm Distance (c) Legend

Figure 2: Distance to Alarm from Mobile Client

tile (the tile numbered 13), and eight tiles surrounding the inner most tile (numbered as 7, 8, 9, 12, 14, 17, 18 and 19, as shown in Figure 3(a)). The mobile client then creates an *internal system defined alarm* over the tile numbered 13 such that whenever the mobile client moves outside the tile 13, an additional tile fetch request will be issued to the map server. For instance, when the mobile client moves out of the inner most tile numbered 13 to the tile 14 as shown in Figure 3(b), additional tile fetches (10, 15 and 20 in the example) are performed, the old *internal system alarm* set on the tile 13 is removed and and a new *internal system alarm* is set on tile 14 to monitor whether the mobile client moves outside the current tile numbered 14. Note that we reuse the very same idea of spatial alarm to serve as a system level monitor to control when to fetch the new tiles from the map service. Choice of tile size depends on several factors. On one hand, the larger the tile size is, the higher the energy conservation will be since the client will be more likely to be further away from the *internal system alarm* on the current tile. On the other hand, the larger the tile size is, the more demand will be set on the client storage capacity. Thus a proper setting of tile size needs to trade off between the storage constraint and the energy conservation need. The good news is that even assuming low storage availability of 2 MB, detailed road map [2] of a medium size city, such as Atlanta or Washington DC, can be stored without having to request additional tiles from the map server. In the rest of this paper we assume that the available storage is limited and hence the tile size is small. We consider the Georgia Institute of Technology campus in Atlanta (which is about $1.61km^2$) to be contained in its entirety in a tile as shown in Figure 3.



(a) Tiled Midtown Map (Client at Tile 13) (b) Client Movement from Tile 13 to Tile 14

Figure 3: Mobile Client Map Window: Tiling of maps to reduce storage costs

4.2 Speed Estimation Methods

Maximum Speed: The use of maximum travel speed of the mobile client has a number of advantages and disadvantages. On the brighter side, one can set the 'Maximum travel speed' by pre-configuration based on either the nature of the mobile client (such as a car on the move or a pedestrian walking on the street), or depending on the types of roads used, or the environmental limitations on the mobile client such as entered a building. For instance, if the mobile client is always expected to be attached to a car then the maximum speed of the car or the highest speed limit on the road on which the car travels can be used as the 'Maximum travel speed'. However, using the maximum travel speed as the speed estimation technique is often over pessimistic since it cannot adapt to the situation where a mobile client may stop for an extended period of time or may suddenly turn onto a road with very low speed limit.

Expected Speed: In most cases the speed of the mobile client will often be lesser than the maximum speeds. This is especially true for road networks that have different traffic patterns at different times of a day, or that have winding roads or other unexpected conditions. Hence, a more pragmatic approach is to estimate the expected speed of the mobile client and use the expected speed measure to determine the safe period (the time to sleep) before the mobile client needs to wake up the next time. In the first spatial alarm middleware prototype system, we calculate the current expected speed using the Exponential Weighted Moving Average (EWMA) scheme (Equation 2) based on current and previous location of the mobile client and the previous expected speed estimate. In addition to the current expected speed weighted by α, the future expected speed is increased

by performing a weighted average $(1-\alpha)$ with the maximum speed to ensure higher guarantee of alarm delivery and zero or lower alarm misses (Equation 3). The lower the α value is, the better the alarm delivery guarantee will be, but the higher the incurred energy cost. When $\alpha = 0$ this function would return the *Maximum Speed*. When $\alpha = 1$ this function would return the EWMA speed. In the experiments we set $\alpha = 0.5$ and $\beta = 0.8$.

$$v^p_{expected} = 0 \tag{1}$$

$$v^c_{expected} = \beta * \frac{D(l_c, l_p)}{t_c - t_p} + (1-\beta) * v^p_{expected} \tag{2}$$

$$v_{expected} = \alpha * v^c_{expected} + (1-\alpha) * v_{max} \tag{3}$$

where $v^p_{expected}$, $v^c_{expected}$, $v_{expected}$ are the previous, the current, and the future *expected* travel speed of the mobile client respectively, t_c and t_p represent the current and previous time instances, l_c and l_p represent the current and the previous location of the mobile client at time instances t_c and t_p respectively and D is the distance function.

4.3 The Aperiodic Wake-Up Algorithms

We have described the role of maximum speed and the calculation of expected speed, and two alternative approaches to measuring the distance from a mobile client to all her spatial alarms based on Euclidean distance and road network distance respectively. In this section, we introduce the concept of safe period based on the distance function and the speed function, and present four safe period based wakeup algorithms by combining the two distance functions and the two speed functions.

$$T_{sleep} = \min(d_{A_1}...d_{A_i}...d_{A_n})/v_{max} \tag{4}$$

$$T_{sleep} = \min(d_{A_1}...d_{A_i}...d_{A_n})/v_{expected} \tag{5}$$

$$T_{sleep} = \min(Rd_{A_1}...Rd_{A_i}...Rd_{A_n})/v_{max} \tag{6}$$

$$T_{sleep} = \min(Rd_{A_1}...Rd_{A_i}...Rd_{A_n})/v_{expected} \tag{7}$$

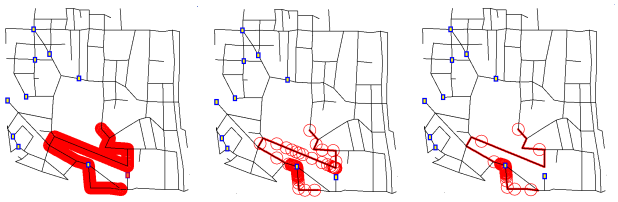
where T_{sleep} is time duration for which the mobile client can sleep without potentially missing delivery of any alarm, n is the total number of alarms installed on the mobile client, d_{A_i}, Rd_{A_i} are the euclidean and road network distances from the mobile client's current location to the i^{th} spatial alarm A_i $(1 \leq i \leq n)$ and v_{max}, $v_{expected}$ are the maximum and expected travel speeds of the mobile client.

Safe Distance with Max Speed: This wakeup algorithm defines the safe distance of a mobile client to each of her spatial alarms by combining the *Euclidean Distance* function and the maximum speed as shown in Equation 4. For any given mobile client, a safe period to all her spatial alarms is the time duration in which the spatial alarm middleware on the mobile client can sleep. We estimate the safe period using the minimum distance from the mobile client to all her spatial alarms divided by the maximum travel speed of the mobile client. At each wakeup, the mobile client will perform two types of tasks. First, it will process the spatial alarms installed on her client device. We will discuss the spatial alarm processing strategies in Section 5. Second, it will estimate the safe period (i.e., the time to sleep), denoted by T_{sleep}, before the next wakeup using equation 4. This wakeup algorithm uses the maximum travel speed to estimate the safe period and thus guarantees that no spatial alarms will be missed no matter how the mobile client changes her movement patterns, such as the number of stops, the length of each stop, or the type of roads traveled. Obviously this wakeup algorithm is over pessimistic in defining the safe period.

Safe Distance with Expected Speed: Clearly, when a mobile client stops for a fairly extended duration, or when the mobile client travels on a low speed limit road for extended period of time, the safe period computation using the maximum travel speed of the mobile client would not be able to take advantage of different opportunities to optimize the safe period. One way to overcome this problem is to use the expected speed as defined in Section 4.2. Since the 'expected speed' would be ideally lesser than the maximum speed, the number of device wakeup can be further expected to be reduced. However, it is possible to miss alarms and no guarantees like in the *Safe Distance with Max Speed* can be offered. Nevertheless, the user can choose to use this approach as opposed to the previous approach when energy of the device is low and the alarm delivery is not critical but beneficial.

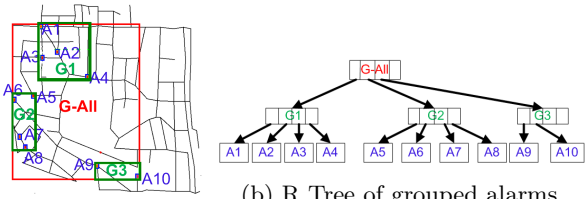
Safe Road Distance with Max Speed: The two Euclidean distance based safe period wakeup algorithms – *Safe Distance with Max Speed* and *Safe Distance with Expected Speed* – clearly outperform the naive periodic wakeup algorithm. However, the use of Euclidean distance function fails to consider the limitations imposed on the movement of mobile clients by the underlying road network. Especially, in an urban scenario, the road network plays a very important role in determining how quickly a mobile client will reach her nearest spatial alarm. We argue that in many cases the road network distance will offer significantly higher accuracy in terms of safe period estimation than the Euclidean distance. Thus using the road network distance function to replace the Euclidean distance function in the safe period calculation allows the mobile client to sleep for a longer duration while ensuring no miss of alarms as shown in Equation 6.

Safe Road Distance with Expected Speed: Sim-



(a)Periodic (b)SD Max (c)Road SD Max

Figure 4: Comparison of Wake Up Strategies



(a) Alarm Grouping (b) R Tree of grouped alarms

Figure 5: Alarm Grouping by Spatial Proximity

ilar to the problem with safe distance with maximum speed algorithm discussed under 'Safe Distance with Expected Speed', the problem of the safe road distance with maximum speed algorithm is the lack of flexibility to take advantage of different movement patterns of the mobile client to optimize the safe period, since the use of maximum speed considerably underestimates the safe period in which the mobile client may sleep. Equation 7 represents the road network distance version.

Figure 4 provides an example scenario to illustrate safe distance and safe road distance based wakeup algorithms with the naive period wakeup approach. Circles in all three cases represent wakeups of the mobile client and rectangles represent the spatial alarms installed on the mobile client. The spatial alarm layout on the road network is shown in all three scenarios. Figure 4(a) represents the *periodic Wake-Up* strategy. The shaded path consists of many circles one overlapped with another, showing the high frequency of the wakeups in this case. Figure 4(b) shows the case of *Safe Distance with Max Speed*. It clearly demonstrates the significant reduction in the number of wakeups comparing to the *Periodic wakeup* case. Figure 4(c) represents the case of *Safe Distance with Expected Speed*. It shows a further reduction on the frequency of the wakeups compared to the safe road distance with maximum speed in Figure 4(b).

5 Minimizing Alarms Checked

In this section we argue that the approach of *check all alarms upon each wakeup* is naive and wasteful of resources. We describe two approaches to minimize the number of alarm checks per wakeup and show how these approaches can reduce the computation cost and the energy consumption involved in alarm checks. In the first approach, we group spatial alarms that are in close spatial proximity, in a hierarchical fashion. Alarm Checking happens in groups, thus minimizing the overall number of alarm checks performed per wakeup. In the second approach, we divide the geographical area of interest into Voronoi regions based on Euclidean distance to the alarms with each region storing information that can quickly identify the nearest alarm in the vicinity. Upon each wakeup, alarm checks are performed only against the 'nearest' alarm by looking up the information in the Voronoi region in which the mobile client currently resides. We also consider using *Network Voronoi Diagrams* as an alternative.

5.1 Hierarchical Grouping of Alarms

When the geographical area in which a mobile client installs her alarms is big, the number of alarms installed is large and distributed across the entire area of interest, checking all alarms upon each wakeup is not only unnecessary but also a clear waste of resources. We first propose to group spatial alarms based on their spatial proximity and check the alarms in selected groups upon each wakeup. The grouping process proceeds in two steps. First, all alarms need to be divided into groups based on spatial proximity with each group associated with a spatial region. Only when the mobile client moves into the region marked by a group, the spatial alarms/subgroups within that group will be checked, and all other alarms/subgroups belonging to the other groups are eliminated from alarm checking, leading to significant saving in terms of computational cost and energy. For instance, Figure 5(a) shows a map of Georgia Tech with a total of 10 alarms installed on a mobile client. We group them into three groups with group one consisting of A_1, A_2, A_3, A_4, group two consisting of A_5, A_6, A_7, A_8, and group three consisting of A_9 and A_10 (see the three innermost rectangles). All these three groups together form the fourth group (see the outer most rectangle in Figure 5(a)).

We use the R Tree [14] algorithm to perform the alarm grouping in a hierarchical fashion as shown in Figure 5(b). Upon each wakeup the mobile client traverses down the tree using her current location and terminates when one of the following conditions becomes true: (i) a leaf node (alarm) is reached; or (ii) the mobile client's location is not bounded by any child's Minimum Bounding Rectangle (MBR). The condition (i) signals that an alarm is satisfied and the appro-

priate action is triggered. This approach reduces the number of alarms to be checked from O(n) in the naive approach of check-all per wakeup to $O(\log_b n)$, where n represents the total number of alarms installed on the mobile client and b represents the minimum number of alarms in a group. In the example shown in Figure 5, $b = 2$ and $n = 10$.

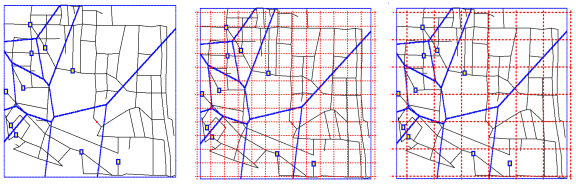
The R-Tree based alarm grouping algorithm is effective in terms of energy saving and resource usage in general and especially it can handle well, the situations where the mobile client continuously adds new spatial alarms into the client middleware system as she moves on the road. However, if the number and the location of the spatial alarms remain unchanged for long duration of time, we can utilize the Voronoi diagram to devise a more efficient alarm group algorithm. We below present two such algorithms, one uses Voronoi regions, called nearest alarm check algorithm, and the other uses the Network Voronoi diagrams, called the road network nearest alarm check algorithm.

5.2 Checking Nearest Alarm Only

The *checking nearest alarm only* algorithm is suitable for the scenarios where the number and location of alarms remain unchanged for long duration of time and no addition or removal of alarms are issued by the mobile client. The *Nearest Alarm Only* optimization consists of two phases. In the first phase, the two dimensional geographical area of interest is divided into grid cells of equal size. Then the Voronoi diagram is overlaid on top of the grid with Voronoi Regions [3, 12] such that each Voronoi Region has a single nearest alarm, as shown in Figure 6(a). To facilitate the search for the nearest alarms for a given mobile client location, we build a grid cell based dense index, in which each cell contained in a Voronoi region will point to the spatial alarm of that region, and each cell that overlaps with k Voronoi regions $(1 < k < n)$ will contain k spatial alarms, each corresponding to one of the k Voronoi regions.

Given a geographical area of interest, such as the state of Georgia or the greater area of Atlanta, there are many mobile clients who will install their personalized spatial alarms. Hence, a third party Voronoi diagram service provider can be used to generate the Voronoi diagram for the entire geographical area, and each mobile client can download the resulting Voronoi diagram for the area of interest on demand.

In the second phase, upon wakeup the mobile client uses her current location to locate the grid cell in which she resides and it takes only O(1) to lookup the nearest alarm in the case where the grid cell of the client is contained in a Voronoi region. In the situation where the mobile client is at boundaries of k Voronoi



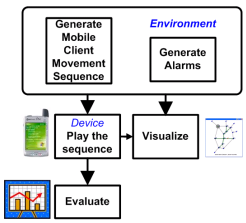
(a) Voronoi Regions (b) Small grid cells (c) Large grid cells

Figure 6: Alarm Grouping by using grid cells

regions $(1 < k < n)$, the grid cell in which the client resides will point to k spatial alarms, all are qualified to be the 'nearest' alarms. In this scenario all the Voronoi regions overlapped with the current location of the mobile client need to be considered, and the alarm check will be performed against the 'nearest' alarm in each of these overlapped Voronoi regions. Clearly, this approach greatly reduces the time to lookup the relevant alarms to be checked, although it is only applicable in the specific scenarios where alarms are not frequently removed or added (since computing the Voronoi diagram for the entire geographical area of interest each time when a new alarm is added or an existing alarm is removed can be quite expensive).

However, the storage requirement for such a scheme is high. Let m denote the total number of grid cells for the entire grid of the geographical area of interest. We construct a dense index with m entries. The storage cost is of the order $O(m)$. With a high value for m, the alarm lookup time takes $O(1)$ but the storage cost will be at $O(m)$ (see Figure 6(b)). One way to reduce the number m of grid cells is to increase the grid cell size at the cost of increased probability of having more than one alarms qualifying as the nearest alarm for a given grid cell (see Figure 6(c)). This is especially true when the grid cell in which the mobile client resides overlaps with two or more Voronoi regions. The larger the size of individual grid cells is, the larger the number of alarms need to be checked at each wakeup. At one end of the spectrum, if there is only one cell covering the entire area of interest, then this grid cell dense index approach simply degrades to the naive approach of check-all alarms upon wakeup. We promote the use of storage to trade for fast computation since this approach offers better energy conservation when alarm addition and deletion is infrequent.

An alternative way to improve the storage cost is to use the Network Voronoi Diagrams [9, 13] instead. The checking nearest alarm with road network algorithm consists of two phases. In the first phase, we need to build network Voronoi diagram that can partition the road network among the alarm nodes. In the second phase, each of the nodes on the road net-



Simulator Design

Parameter	Default
Number Of Alarms (N_A)	10
Clustering Factor	2
Simulation Duration (T_t)	3 hrs
100% CPU Power (P_{100})	540 mW
IdlePower (P_i)	100 mW
SleepPower (P_s)	8.39 mW
Minimum Wake Duration (T_m)	0.001 hrs
Battery Capacity (C_b)	2052mWh

Simulator Parameters

Figure 7: Simulator Design

work graph is associated with a list of 'nearest alarms' based on the road network distance. Thus the alarm checking operation is performed using the O(Nd) algorithm for constructing Network Voronoi Diagrams, where N is the number of nodes (vertices) in the road network graph and d is the maximum degree of any vertex in the graph. Using the road network approach greatly minimizes storage costs as we can bring down the storage cost from O(Nd) to O(N) for the N number of nodes in the road network graph.

6 Experimental Evaluation

We have described the four wakeup algorithms and the three alarm check algorithms. In this section we evaluate our architecture, the wakeup algorithms, and the alarm check algorithms in terms of (a) total energy consumed, (b) total battery lifetime, (c) alarm density and alarm distribution, and (e) alarm delivery quality in terms of alarm misses. We below first describe briefly our simulator design and our experimental setup. Then we present the experimental results, demonstrating the effectiveness of our proposed middleware architecture for energy-efficient processing of spatial alarms.

6.1 Experimental setup

In order to realistically simulate the spatial alarm middleware system on the mobile clients we develop a simulator that has the environment and work load generation modules completely separated out from the specific wakeup and alarm check algorithms used by the mobile clients. Figure 7 shows a sketch of the key components of the simulator. The *environment module* uses the supplied map and generates mobile client position trace for the duration of time specified. It also generates the list of alarms based on the alarm distribution defined by the *Alarm Clustering Factor* (see the next subsection for details). The *Device module* is independent of the *environment module* and can be configured for different mobile clients with different combinations of the Wake-Up algorithm and the Alarm Check algorithm. Depending on the settings given by each mobile client, it *plays* the client's position trace file along with the *alarms file* generated by the *environment module* and records the following statistical information: (a) Number of wakeups (b) Number of alarm checks (c) Number of alarms delivered (d) Storage cost, apart from the information required for visualizing the behavior of the system, such as those shown in Figure 4. The *Evaluation* module comprises scripts to systematically vary environmental parameters listed in the simulator parameter table given in Figure 7, which generate performance graphs presented in the rest of Section 6.

In reality the spatial alarms installed by a mobile client are likely to be restricted in certain regions of interests, such as a few miles around work place or home. In order to simulate such behavior we introduce the concept of *Clustering Factor*. We simulate the distribution of spatial alarms of a mobile client using various clustering factors. In general, a clustering factor of m implies that the alarms are distributed over only $\frac{1}{m}^{th}$ of the map.

6.2 Estimating Energy and battery life

In order to measure the energy consumed and battery lifetime for various wakeup algorithms and alarm check algorithms, we use the device energy values corresponding to the itsy pocket computer [4] given in the energy parameter table of Figure 7 as our reference model. In the rest of this subsection we show how one can express the total energy E_t consumed as a function of the number of wakeups (N_w), the minimum time duration per wakeup (T_m), the total time duration (T_t), the power consumption of the mobile device while awake (P_{100}), idle (P_i) and asleep (P_s), and the *Alarm Check Ratio* (C_r) which represents the ratio of the number of total actual alarm checks performed to the maximum total checks that can be performed $N_{A_{MAX}}$ during the minimum wakeup durations T_m.

First, we compute the total energy E_t as the sum of the energy spent awake and the energy spent in the sleep state. Let T_w and T_s denote the time periods of a mobile client in awake and asleep state respectively and P_w and P_s denote the energy spent per time unit in awake and asleep state respectively. The following equation holds.

$$E_t = T_w P_w + T_s P_s \tag{8}$$

Let N_w denote the number of wakeups and T_m denote the minimum wakeup duration. We can compute the time duration for which the mobile client is awake using the following formula:

$$T_w = N_w \times T_m \tag{9}$$

Similarly, the time duration for which the mobile client is asleep, denoted by T_s, as the difference of total time duration T_t and the wakeup time duration T_w:

$$T_s = T_t - N_w T_m \tag{10}$$

The energy required for keeping the device (mobile client) awake is dependent on whether the mobile client is computing or not. If there are more alarms that the client has to compute for longer duration of time, then the power consumption will be higher. After the mobile client finishes checking for the alarms at each wakeup, it goes to the idle state for a short duration before going to sleep. Let P_i denote the power spent by the mobile client during an idle state. A lower value of *Alarm Check Ratio* indicates that fewer alarms need to be checked, and hence higher the energy conservation. Thus, we can compute the power consumed during wakeup, denoted as P_w, by the following formula which has two components - one to incorporate the power consumed while awake and the other while idle before going to sleep.

$$P_w = C_r P_{100} + (1 - C_r) P_i \tag{11}$$

Where C_r denote the ratio of 'total alarm checks performed' to the 'maximum number of alarms checks that can be performed before the device has to go to sleep'. It can be defined in terms of the total number of actual alarm checks (N_C), the maximum total number of alarm checks that can be performed ($N_{A_{MAX}}$) in the minimum wakeup duration, and the number of wakeups (N_w):

$$C_r = \frac{N_C}{N_{A_{MAX}} * N_w} \tag{12}$$

By substituting Equations 9, 10, and 11 in Equation 8, we obtain the *Total Energy consumed* as follows:

$$E_t = N_w T_m (C_r P_{100} + (1 - C_r) P_i) + (T_t - N_w T_m) P_s \tag{13}$$

Let C_b be the battery capacity, T_t be the total time of the experiment, and E_t represent the energy spent during T_t. The battery lifetime of a mobile client, denoted by T_b, can be calculated by the following equation:

$$T_b = \frac{C_b \times T_t}{E_t} \tag{14}$$

6.3 Experimental Results

In this section we present a set of experimental results. Our results demonstrate three important conclusions. First, the proposed wakeup and alarm check algorithms offer significant (up to 6.4 times) reduction in terms of energy consumption in comparison to the naive approach with periodic wakeups followed by checking all alarms per wakeup. Second, the *Safe Road Distance with Max Speed* wakeup algorithm offers the maximum energy conservation with 100% alarm delivery guarantee. Third but not the least, the *alarm grouping* check algorithm is the most flexible among the alternative alarm check strategies and offers significant reduction in terms of energy consumption, and significant improvement (up to 50%) in battery life when the number of alarms is high. The *Nearest-only Alarm* and the *Network Nearest-only Alarm* algorithms offer even better improvements in terms of energy consumption but have limited applicability and can only be used in those cases where addition and removal of alarms are less frequent and low in numbers.

6.3.1 Effect on Overall Energy Consumption

Figures 8, 9, 10 compare energy and battery life time for a mobile device by varying the number of alarms per map tile for different wakeup algorithms (Figure 8), different alarm check algorithms (Figure 9), and five alternative combinations of wakeup algorithm and alarm check algorithm (Figure 10). We used the measurements for the Itsy Pocket Computer [4] given in the energy parameter table of Figure 7 for these experiments. The results were plotted using the equations in Section 6.2. Apart from the values listed in the table in Figure 7, we want to demonstrate how the algorithms perform when the number of alarms reaches high values and assume that the mobile client can check only 100 alarms ($T_{A_{MAX}}$) in 0.001 hrs. Though in reality this number might be higher, by making such a pessimistic assumption we get to stress-test the system and see its behavior at the boundary conditions.

Figure 8 (left and center) compare the energy consumption of the individual wakeup algorithms by fixing an alarm check algorithm and we use the naive alarm check algorithm in this case. It shows that our wakeup algorithms (especially the Safe Road Distance algorithms) perform close to 6 times better than the naive algorithm of periodic wakeup and check all alarms per wakeup. The increase in energy consumption is almost linear with the increase in the number of alarms for all four of our wakeup algorithms because we use the naive alarm checking strategy in all the cases. Figure 8 (right) shows the corresponding lifetime of the battery for each of the wakeup algorithms. As shown in the figure, all our wakeup algorithms offer significant improvement in terms of the battery lifetime of the mobile client and the battery lifetime converges when the number of alarms are high, indicating that by using the wakeup algorithms alone, it is not

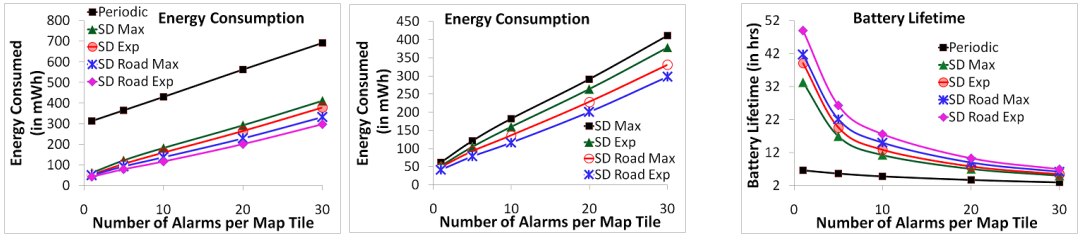


Figure 8: Performance of WakeUp Strategies

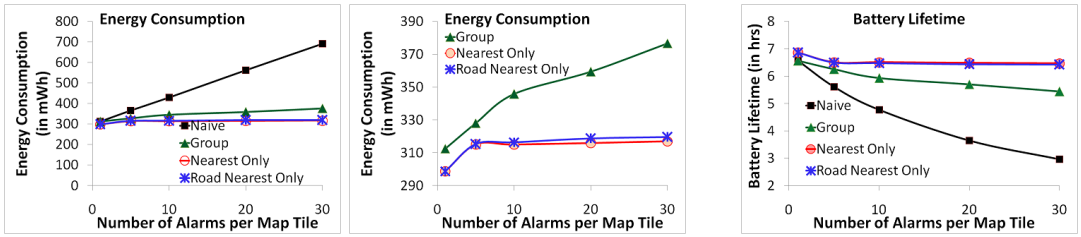


Figure 9: Performance of Check Strategies

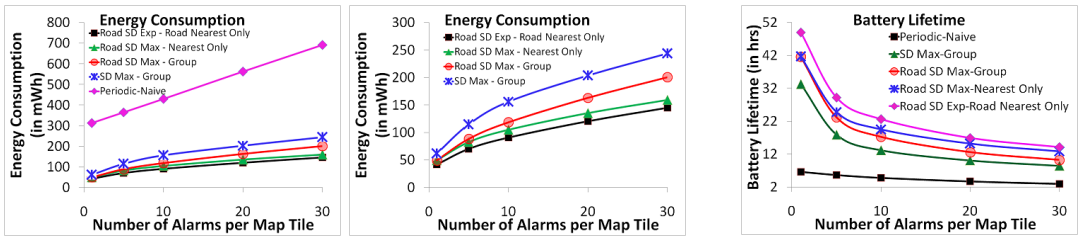


Figure 10: Performance of WakeUp-Check strategy combinations

always sufficient to reduce the energy consumption involved. For such scenarios, alarm check algorithms are necessary for energy conservation purpose.

Figure 9 (left and center) compare the energy consumption of the alarm check algorithms by fixing the wakeup Strategy (Periodic WakeUp Strategy is used in this case). The graphs show that all three proposed alarm check strategies turn out to outperform the naive approach that checks all alarms upon each wakeup, especially when the number of alarms are high. The Energy consumption with the naive check algorithm grows almost linearly while with our proposed alarm check algorithms, the increase in terms of energy consumption is much slower. This is to be expected because the naive approach checks all alarms upon each wakeup, which can be significantly degraded when the number of alarms is high. Figure 9 (right) shows the corresponding effect on the battery lifetime. The flexible alarm grouping based checking algorithm offers close to 86% improvement over the conventional approach. Also, interesting to note is that the 'Road Nearest' based check is almost as energy efficient as the 'Nearest only' check algorithm even though it uses far lesser storage (as shown in Figure 14(center-left)). Further, the trend shows that these improvements grow with the increase in the number of alarms.

Finally, we examine the effectiveness of the wakeup and alarm check combinations. Figure 10 shows some of the interesting combinations. Due to the space constraint, we omit for a large number of possible combinations. The battery lifetime plot in Figure 10 (right) is similar to Figure 8 (right). Even at higher number of alarms per map tile, we show that the battery lifetime of a mobile client is considerably better in the wakeup case, reflecting the need for a good alarm checking strategy that parts from the wakeup algorithms.

Figure 11(left and center-left) compare the variation of Wake-Up frequencies with the increase in number of alarms. As one would expect the Periodic Wake-Up strategy is indifferent with the increase in the number of alarms. When the mobile client has only one alarm per map tile, the proposed wakeup algorithms reduce the number of wakeups by almost an order of magnitude. The two *Safe Road Distance* algorithms perform better than the *Safe Distance* algorithms. The wakeup algorithms that use *expected*

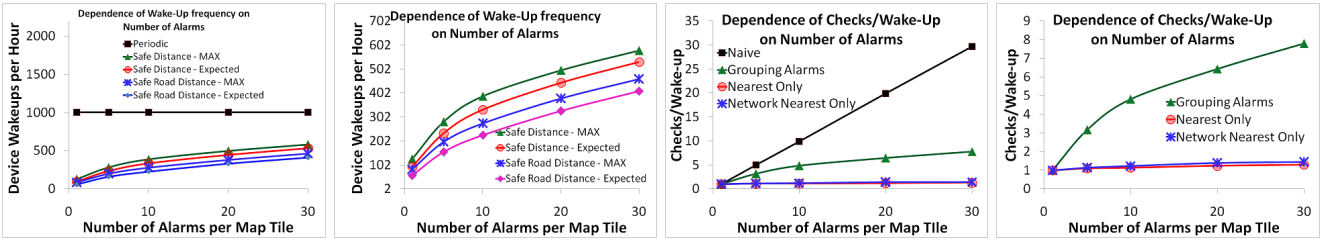


Figure 11: Effect of Changes in Number of Alarms

speed reduce the frequency of wakeups further. However, as shown in Figure 13 the *expected speed* algorithms may not guarantee the delivery of all alarms at all times. Also all the proposed wakeup algorithms increase the wakeup frequency as the number of alarms increases and similarly, all the check algorithms increase the number of alarm checks as the number of alarms increases. This is expected because with increase in number of alarms the average minimum distance to any alarm would be reduced and hence both the *Safe Distance* and *Safe Road Distance* Wake-Up algorithms will allow the mobile client to sleep only for shorter durations. Note that in this set of experiments we use a *clustering factor* of two and all the alarms are limited to one half of the map, resulting in the frequency of wakeups to level out for high number of alarms.

Figure 11(right and center-right) compare the Alarm Check algorithms as the number of alarms varies. As one would expect, with increase in number of alarms, the number of checks per wakeup also increases. Such an increase is linear for the *naive approach* that checks against all the alarms at each wakeup. However, for the alarm grouping approach, which uses a R-Tree [14] based lookup, the increase is sub-linear. The *Nearest Alarm Only* and the *Nearest Road Network Distance Alarm Only* offer almost constant number of checks at each wakeup irrespective of the number of the alarms installed. In reality, with increase in number of alarms, the number of checks for these two strategies can also go up but is almost negligible in practical scenarios. Only in some extreme cases where almost all the alarms are overlapping with each other, these strategies would end up requiring checks on all the alarms at each wakeup.

Figure 12(left and center-left) show the performance comparison of periodic wakeup with the four proposed wakeup algorithms with an increase in the Clustering Factor. We see that as the Clustering factor increases, the Wake-Up algorithms tend to perform better. This is simply because when the clustering factor is high, it implies that the alarms are clustered around a portion of the map, thus the minimum distance to the nearest alarm from the client's current

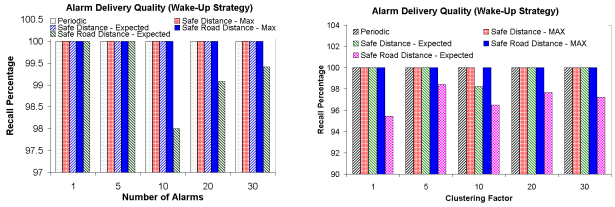


Figure 13: Alarm Delivery Quality

location is higher, and the client has longer time to sleep.

Figure 12(right and center-right) show the effect of changing clustering factor on the Alarm Check Strategies. It is interesting to note that with increase in clustering factor, the *grouping near by alarms* algorithm performs particularly well in terms of performance improvement. This is because the closer the alarms are distributed, the smaller the size of the Minimum Bounding Rectangle (MBR) required to enclose them, and hence the lesser the number of alarm checks to be performed at each wakeup. It is also important to note that at higher clustering factors there is a slight increase in the number of alarm checks for the *Nearest Alarm Only* algorithm. This is because when alarms get very close to each other, *grid cells* used by this strategy tend to have more than one 'Nearest' alarms which in turn result in checking of more than one alarms for satisfaction.

6.3.2 Alarm Delivery Quality

Figure 13 evaluates the quality of the alarm delivery provided by each of the wakeup algorithms with variations in the *Number of Alarms* and the *Clustering Factor* respectively. *Periodic*, *Safe Distance with Maximum Speed* and *Safe Road Distance with Maximum Speed* perform perfectly in terms of 100% delivery of alarms. However, *Safe Road Distance with Expected Speed* misses a few of alarms. Note that in some cases only about 95% of the alarms are delivered. One would expect the same behavior for the *Safe Distance with Expected Speed* approach. But surprisingly in most of the cases the *Safe Distance with Expected Speed* ap-

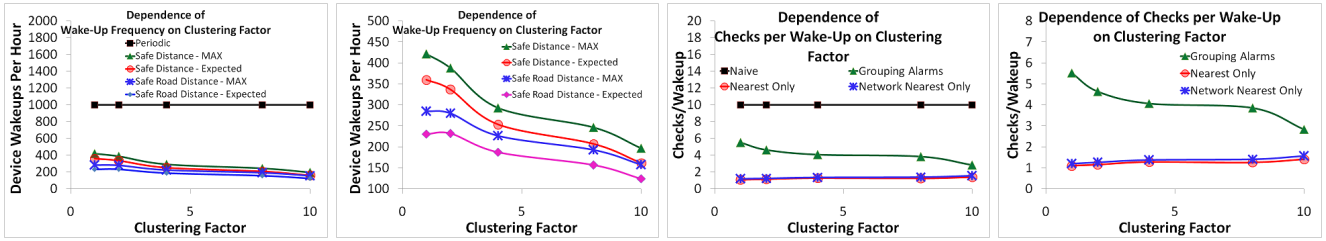


Figure 12: Effect of Clustering of Alarms

proach manages to have a 100% alarm recall rate. We believe that this is because the use of *Safe Euclidean Distance* played a critical role in offsetting the errors produced in expected speed estimation. This experiment shows that the *Expected Speed* algorithms perform better in terms of reducing the number of wake-ups and saving energy. However they are not flawless in delivering alarms with 100% recall, and hence should be used with care. It is important to note that the alarm check algorithms do not affect the alarm delivery quality. Due to the space constraint we omit the corresponding experimental results in this paper.

6.3.3 Effect of Alarm Group Size

Figure 14(left) shows the effect of the alarm group sizes on the final computation required for spatial alarm processing. A maximum alarm group size of m implies that the maximum fan out of the R Tree [14] used to group the alarms is m. In other words, no more than m alarms can be put together to form a group and no more than m of such groups can be put together into a higher level group, and so forth. It is interesting to note that the larger the group size value of m is, the higher the number of alarms will need to be checked, and hence worse the performance of alarm process will be. This is particularly the case when the alarms are scattered all around the map ($CF = 1$).

6.3.4 Storage Cost for Nearest Alarm Check

We measure the tradeoffs between the *Storage* requirement vs *minimizing number of checks* for the *Nearest Alarm Only* Alarm Checking algorithm. In order to make the effects more pronounced we used forty alarms (on the same map tile of about $1.61km^2$ area) with a clustering factor of three such that a large number of alarms would be clustered around a small region. Our experimental results show that with increase in number of *grid cells* per row of the map tile, the storage requirements for the dense index also increase rapidly (Figure 14(center-left), while at the same time the number of checks required at each wakeup is reduced at a very high rate (Figure 14(right)). However, the *Network Nearest Alarm Only* approach is more scalable as shown in Figure 14(center-right and center-left).

Algorithm	Battery	Lines of Code	Recall
Periodic	5.93 hrs	14	100%
SD Max	13.11 hrs	24	100%
SD Exp	14.84 hrs	55	100%
SD Road Max	17.26 hrs	410	100%
SD Road Exp	20.02 hrs	451	98%

Table 1: Wake-up Strategy Comparison (10 alarms case)

Algorithm	Battery	LOC	Storage growth
Naive	2.96 hrs	41	constant
Group	5.44 hrs	567	linear
Nearest	6.47 hrs	132	quadratic
Road Nearest	6.42 hrs	359	linear

Table 2: Check Strategy Comparison (30 alarms case)

Finally, Table 1 and Table 2 compare the different algorithms along several different dimensions. 'Storage growth' denotes the growth in storage requirements in addition to the list of alarms. Clearly, there is a trade off involved between simplicity (LOC), efficiency (battery), quality/recall (for wake-up algorithms) and storage (for check algorithms). Depending on the intended usage scenario, the system implementation could choose between the different algorithms to maximize the utility.

7 Related Work and Conclusion

The idea of location based reminder and its use have been discussed in several human-computer interaction projects in the recent years [18, 23, 7, 17, 5, 22]. Most of them primarily concentrate on usability of such location reminder systems from the human-computer interaction point of view. Certain others like [16] focused on identifying the user's *logical* location by mapping information such as IP Address, latitude and longitude information in GPS readings. The logical location could then be used to automatically connect the mobile client to local resources such as printers. On the other hand, there are several other works like [11],

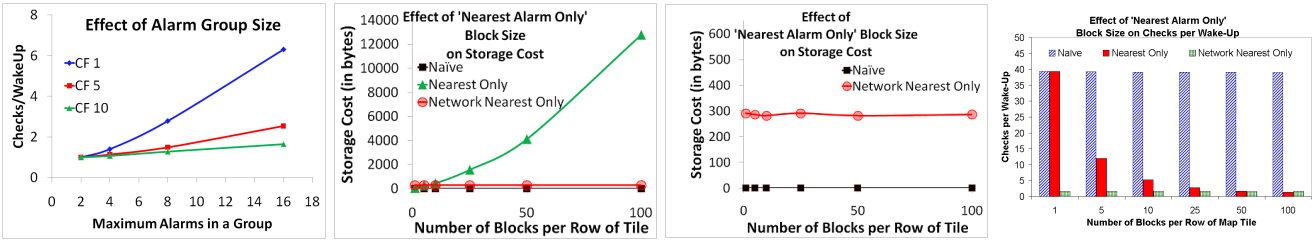


Figure 14: Effect of Alarm Group Size and Storage Requirements for the Check Strategies

[10], [19] that are dedicated to general strategies for minimizing energy consumptions on mobile devices.

[21] introduces the idea of *Query Indexing* and *Velocity Constrainted Indexing* solves the more generic problem of continous queries. However, given the generic nature of the problem it does not exploit aspects related to the road network and optimizations such as query grouping.

We have presented an energy efficient framework for processing spatial alarms on mobile clients. Our approach to spatial alarms provides two systematic methods for minimizing energy consumption on mobile clients. First, we introduce the concept of safe period to reduce the number of unnecessary wakeups for spatial alarm evaluation, enabling mobile clients to sleep for longer periods of time. We show that our safe period techniques can significantly minimize the energy consumption on mobile clients compared to periodic wakeups while maintaining the desired accuracy and timeliness of spatial alarms. Second, we develop a suite of techniques for minimizing the number of location triggers to be checked for spatial alarm evaluation at each wakeup. This further reduces the computation cost and energy expenditure on mobile clients. Our experiments show that the proposed client-based spatial alarms architecture offers significant improvements on battery lifetime of mobile clients, while maintaining high quality of spatial alarm services compared to the conventional approach of periodic wakeup and checking all alarms upon each wakeup.

References

[1] Google maps, http://maps.google.com.

[2] U.s. census bureau. tiger, tiger/line and tiger-related products. available at http://www.census.gov/geo/www/tiger/.

[3] F. Aurenhammer. Voronoi diagrams a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[4] J. F. Bartlett, L. S. Brakmo, K. I. Farkas, W. R. Hamburgen, T. Mann, M. A. Viredaz, C. A. Waldspurger, and D. A. Wallach. The itsy pocket computer, 2000.

[5] V. Bazinette, N. H. Cohen, M. R. Ebling, G. D. H. Hunt, H. Lei, A. Purakayastha, G. Stewart, L. Wong, and D. L. Yeh. An intelligent notification system. In *IBM Research Report*, 2001.

[6] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation, August 3, 1993.

[7] A. K. Dey and G. D. Abowd. Cybreminder: A context-aware system for supporting reminders. In *Handheld and Ubiquitous Computing: Second International Symposium, HUC 2000, Bristol, UK Proceedings*, 2000.

[8] E. W. Dijkstra. A note on two problems in connection with graphs., 1959.

[9] M. Erwig. The graph voronoi diagram with applications. In *Networks*, pages 156–163, 2000.

[10] K. Flautner and T. Mudge. Vertigo: automatic performance-setting for linux. *SIGOPS Oper. Syst. Rev.*, 36(SI):105–116, 2002.

[11] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 48–63, New York, NY, USA, 1999. ACM Press.

[12] S. Fortune. Voronoi diagrams and delaunay triangulations. pages 377–388, 1997.

[13] M. Graf and S. Winter. Netzwerk-voronoi-diagramme. sterreichische zeitschrift fr vermessung und geoinformation. pages 166–174, 2003.

[14] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM Press.

[15] J. Hakkila and J. Mantyjarvi. User experiences on combining location sensitive mobile phone applications and multimedia messaging. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 179–185, New York, NY, USA, 2004. ACM Press.

[16] J. Heidemann and D. Shah. Location-aware scheduling with minimal infrastructure. In *USENIX Conference Proceedings*, pages 131–138, San Diego, CA, June 2000. USC/Information Sciences Institute, USENIX.

[17] S. W. Kim, M. C. Kim, S. H. Park, Y. K. Jin, and W. S. Choi. Gate reminder: a design case of a smart reminder. In *DIS '04: Proceedings of the 2004 conference on Designing interactive systems*, pages 81–90, New York, NY, USA, 2004. ACM Press.

[18] P. J. Ludford, D. Frankowski, K. Reily, K. Wilms, and L. Terveen. Because i carry my cell phone anyway: functional location-based reminder applications. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 889–898, New York, NY, USA, 2006. ACM Press.

[19] T. Mudge. Power: a first-class architectural design constraint. In *Computer*, pages 52–58, Michigan Univ., Ann Arbor, MI, USA, April 2001.

[20] J. P. Munson and V. K. Gupta. Location-based notification as a general-purpose service. In *WMC '02: Proceedings of the 2nd international workshop on Mobile commerce*, pages 40–44, New York, NY, USA, 2002. ACM Press.

[21] 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. *IEEE Trans. Comput.*, 51(10):1124–1140, 2002.

[22] N. M. Sadeh, F. L. Gandon, and O. B. Kwon. Ambient intelligence: The mycampus experience. In *CMU-ISRI-05-123*, 2005.

[23] T. Sohn, K. A. Li, G. Lee, I. Smith, J. Scott, and W. G. Griswold. Place-its: A study of location-based reminders on mobile phones. In *UbiComp 2005: Ubiquitous Computing*, 2005.