Mobile Netw Appl
DOI 10.1007/s11036-010-0221-2

An Energy Efficient Middleware Architecture for Processing Spatial Alarms on Mobile Clients

Anand Murugappan · Ling Liu

Abstract Time based alarms are used by many on a daily basis. 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 a middleware architecture for energy efficient processing of 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, 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. 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 spatial alarms middleware architecture offers significant improvements on 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.

Keywords spatial alarms · location based alarms · location based services · mobile applications · energy efficient algorithms

1 Introduction

Spatial alarms are considered by many as one of the critical mobile location-based applications in future computing environments. For instance, a user could set a spatial alarm on her mobile client (e.g., her smart phone or her personalized navigation system in her car), which alerts her whenever she is nearby the dry clean store or the grocery stores in her neighborhood, reminding her to pick up or drop off her dry cleaning items or automatically popping up her grocery shopping list as a spatial reminder. Furthermore, the mobile user can install a spatial alarm targeted at a particular location (e.g., her favorite dry clean store) with a specific music piece or with a voice recording or with an image of the store. Upon her arrival of the alarm region, the music piece or the audio and video will play on her mobile client (e.g., cellphone). One popular way to install such personalized spatial alarms is to use the map software on the mobile client of the user by marking the area of interest as the location alarm area, with associated actions upon the firing of corresponding location triggers. Most of the existing research on spatial alarm type of mobile or wireless

A. Murugappan (✉) · L. Liu
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
e-mail: anandm@cc.gatech.edu

L. Liu
e-mail: lingliu@cc.gatech.edu

Published online: 23 January 2010

applications are produced from the Human–Computer Interaction community, which focuses on the usability study of location reminders on mobile clients or the efficient usage of multimedia to register and notify of location reminders on a mobile client on behalf of the human user of the device. Surprisingly, there are little research to date that is dedicated to energy efficient evaluation of spatial alarms on mobile clients.

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 handheld 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 overview

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. 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.

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 [3] pocket computer spends 540 mW power in the *System Idle, 0% processor idle* state, spends 100 mW power in the *System Idle, 95% processor idle* state, and while in the *Sleep mode* it just spends 8.39 mW 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 [3] have a battery lifetime of only 3.8 h when running in the high energy *System Idle, 95% processor idle* state, while in the *Sleep mode* the lifetime is as high as 279 h.

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.

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.

In this paper 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. 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 3.

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. Thus our phase II optimization focus on minimizing alarm checks with three strategies. We extend three popular indexing algorithms to perform the *alarm grouping optimizations* using R-Tree [13], Voronoi diagram that partitions the two-dimensional coordinate space into Voronoi Regions [2, 11], resulting in a very efficient nearest alarm lookup. To mitigate the additional storage cost, we use the road network information to generate a Network Voronoi Diagram [8], 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 4.

3 Phase I optimization: minimizing device wakeups

Our middleware 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 accuracy 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 the distance of the client to all her alarms and the travel speed of the mobile client. The rest of this section is organized as follows. First, we describe two different techniques for estimating the distance the mobile client's current location to any of her alarms. Second, we describe the maximum speed and the expected speed as two alternative measures to the speed of mobile client. Finally, we present the four optimization algorithms that estimate the time to sleep, or the so-called the safe period, for the mobile client before her next wakeup. Our approach also ensures that alarms can be added and removed while the system is operational and such dynamic alarm addition and deletion do not interfere with the safe period estimation of the mobile client.

3.1 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.

3.1.1 Euclidean distance to an alarm

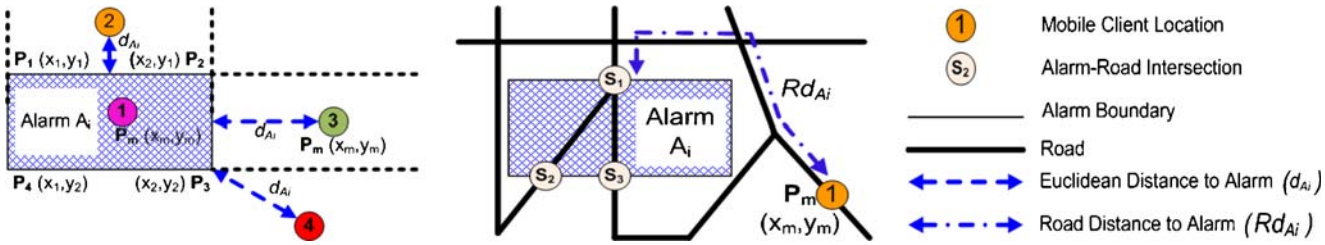
Given a spatial alarm A_i with rectangular spatial alarm region represented by four vertices of the rectangle: (P_1, P_2, P_3, P_4) where $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_1)$, $P_3 = (x_2, y_2)$ and $P_4 = (x_1, y_2)$. Let the mobile client be at P_m represented by the coordinates (x_m, y_m), then 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 Fig. 1a) 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 scope then the distance is the minimum of the Euclidean distances to the four vertices. The four cases can be formally defined as follows:

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

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.

3.1.2 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 [5, 7] to estimate the distance from the mobile client's current location to an alarm as shown in Fig. 1b. 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,



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

Fig. 1 Distance to alarm from mobile client

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 Fig. 1b. 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 [5, 7].

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 Fig. 2. Given the midtown Atlanta map as the area of interest as shown in Fig. 2a. 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 tile (the tile numbered 13), and eight tiles surrounding the inner most tile are numbered 7, 8, 9, 12, 14, 17, 18 and 19, as shown in Fig. 2b. 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, Several factors affect the decision on the tile size. On one hand, the larger the tile size is, the higher the energy conservation will be since the client will 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

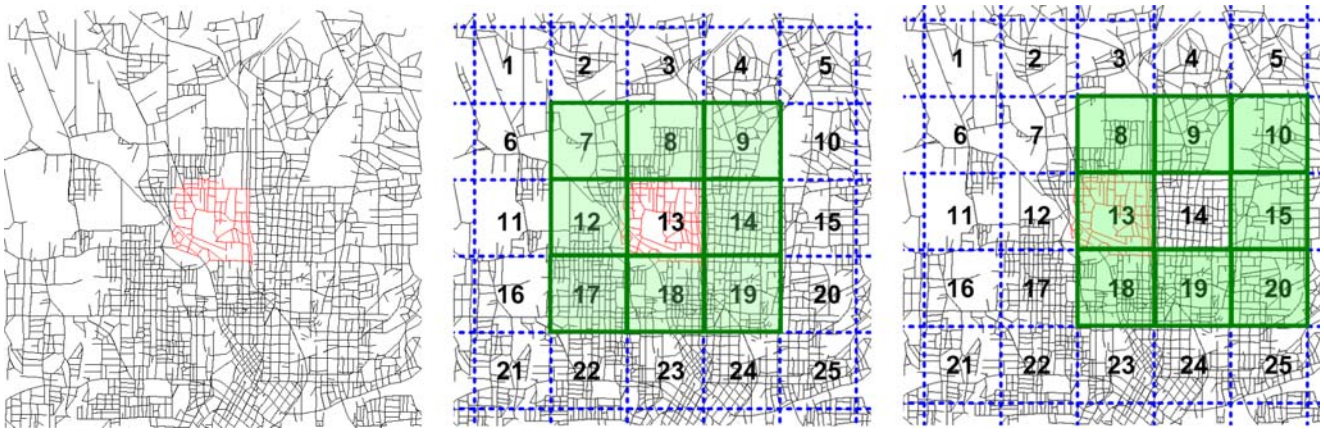


Fig. 2 Mobile client map window: tiling of maps to reduce storage costs

2 MB, detailed road map [1] 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 Fig. 2.

3.2 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: If in most cases the world is more benign, then 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 (Eq. 9) 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 (Eq. 10). 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, and 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.

Note that client travel speed estimation is essential in determining the number of wakeups to be used. Our speed estimation formula does not assume that the mobile client knows its exact travel speed. Instead, we use the current and previous location of the mobile client, her current and previous wakeup time instances, and the previous expected speed to estimate her current and her future expected travel speeds. By taking into consideration the maximum velocity, the current and the past movement behavior, we show that our expected speed calculation presents a reasonable approximation to the future travel speed of the mobile client.

3.3 The four aperiodic wake-up algorithms

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 distance functions with the speed functions.

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

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

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

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

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 $(1 \le i \le n)$ and v_{max},

$v_{expected}$ are the maximum and expected travel speeds of the mobile client as defined below.

$$v^{p}_{expected} = 0 \tag{8}$$

$$v^{c}_{expected} = \beta * \frac{D(l_c, l_p)}{t_c - t_p} + (1 - \beta) * v^{p}_{expected} \tag{9}$$

$$v_{expected} = \alpha * v^{c}_{expected} + (1 - \alpha) * v_{max} \tag{10}$$

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.

The parameter α is a weigh factor that is used to compute the expected velocity of a mobile in the near future by balancing between the current velocity of the mobile and the past maximum travel speed of the same mobile client. The value alpha varies between 0 and 1. If α is set close to 1 then the expected future velocity is computed primarily based on the current velocity of the mobile. If α is set close to 0, then the expected future velocity is computed primarily based on the past maximum-speed of the mobile.

We choose a value that is right in the middle (i.e., 0.5) between the two points. The objective of this α setting is to compute the future expected velocity based on equal weight of both the current velocity and the past maximum velocity of the client. This allows us to predict the future expected velocity of a mobile by combining her current velocity and her maximum travel speed.

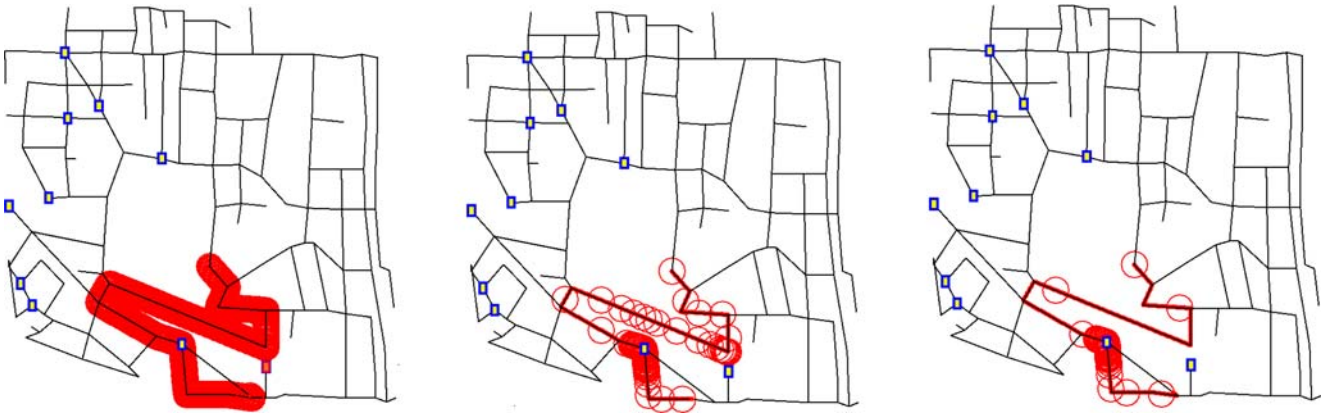
β is also a parameter with value between 0 and 1. It is used as a knob to control how quickly the system adapts to changing speeds. A higher value would result in the system responding faster to changes in speeds while a lower value would result in slower adaptation while placing more weight on the historical values. We chose 0.8 to keep the adaptability relatively high but slightly lower than 1 to be influenced by historical values as well.

Safe Distance with Max Speed 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 Eq. 4. At each wakeup, the mobile client will perform two tasks—(a) process the spatial alarms installed on the client (b) estimate the safe period (i.e. the time to sleep) denoted by T_{sleep}, before the next wakeup. **Safe Distance with Expected Speed** in Eq. 5 exploits the fact that not all mobile clients might travel at the maximum possible speed at all times. Since 'expected speed' would ideally be lesser than the maximum speed, the number of device wakeups can be further reduced at the expense of potentially missing some alarms. **Safe Road Distance with Max Speed** in Eq. 6 exploits the limitations imposed on the movement of the mobile client by the underlying road network while ensuring no alarms are missed. **Safe Road Distance with Expected Speed** in Eq. 7 is a natural extension and combines the advantages of the 'expected speed' approach with the 'road distance' approach.

Figure 3 provides an example scenario to illustrate the 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 3a 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 3b 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 3c represents the case of *Safe Road 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 Fig. 3b. It is interesting to note that some of the wakeups are slightly further apart when the mobile client travels away from the spatial alarms in terms of road network distance. This example also illustrates the contrast between Euclidean distance and road network discussion. For instance, the spatial alarms on the lower left bottom of the map, though close to the mobile client on the move in terms of Euclidean space, are relatively further away in terms of actual road network distance. Hence by using *Safe Road Distance*, a considerable number of unnecessary wakeups can be avoided.

4 Phase II optimization: minimizing alarms checked

The conventional approach is to periodically wakeup the mobile client and to check against **all** the alarms individually upon each wakeup. We have shown that periodic wakeup is naive and consumes a lot of energy unnecessarily. We have shown in the previous section that by utilizing the distance to alarms and the estimate of travel speed of the mobile client, our safe period based wakeup algorithms significantly outperform the periodic wakeup approach in terms of energy efficiency



(a) Periodic (b) SDMax Speed (c) Road SD Exp Speed

Fig. 3 Comparison of wake up strategies

without introducing loss of alarms. In this section we argue that the approach of *check all alarms upon each wakeup* is naive and waste of resources. We describe three 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. In the third approach, we further reduce the storage cost involved with the Euclidean distance based Voronoi diagram approach by dividing the road network map in the geographical area of interest into network distance based Voronoi regions, based on the number of road network nodes, using *Network Voronoi Diagrams*. We below describe each of these three alarm check optimizations in detail.

4.1 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, Fig. 4a 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 Fig. 4a).

We use the R Tree [13] algorithm to perform the alarm grouping in a hierarchical fashion as shown in Fig. 4b. 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 appropriate 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 our running example shown in Fig. 4a, we have $b = 2$ and $n = 10$.

The R-Tree based alarm grouping algorithm is effective in terms of energy saving and resource usage

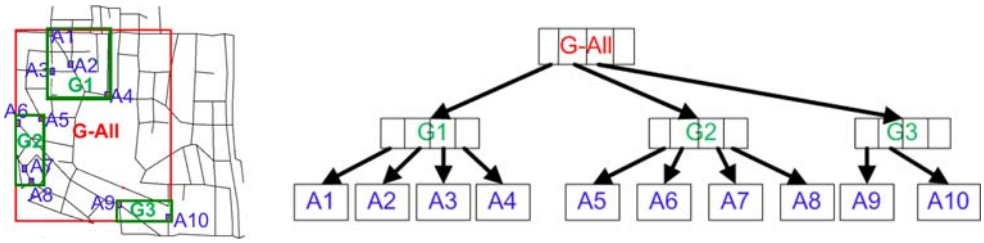


Fig. 4 Alarm grouping by spatial proximity

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

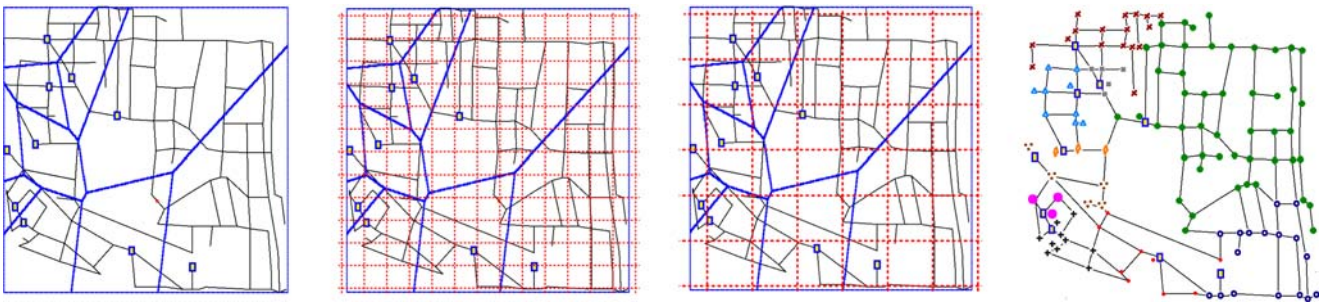
in general and especially it can handle well, the situations where the mobile client continuously adds new spatial alarms into the client middleware system as she moves on the road. However, if the number and the location of the spatial alarms remain unchanged for long duration of time, we can utilize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

4.2 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 [2, 11] such that each Voronoi Region has a single nearest alarm, as shown in Fig. 5a. 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 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 complexity of 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



(a) Voronoi Regions (b) Small grid cells (c) Large grid cells (d) Road Network Voronoi

Fig. 5 Alarm grouping using Voronoi and road network Voronoi approaches

of interest each time when 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 Fig. 5b). 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 Fig. 5c). 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 are infrequent.

4.3 Checking nearest alarm only with road network

We have discussed the cost of storage for Voronoi region based approach. In order to minimize the storage requirement, we consider the usage of the Network Voronoi Diagrams [8, 12] instead without degrading performance significantly.

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 alarm nodes such as each Voronoi region containing one alarm. Figure 5 illustrates the alarm grouping by dividing the road network map into network distance based Voronoi regions. Each alarm (in rectangle shape) has a color associated with it and the vertices of the road network graph for which that particular alarm is the nearest (road network distance) are represented by the same color. Note that there are certain points which are closer to certain alarms in terms of Euclidean distance but not so if Road Network distance is considered. Such alarms in the Road-Network-Alarm-Grouping strategy would be grouped with the Road-Network-Nearest-Alarm and not the Euclidean-Nearest-Alarm.

In the second phase, each of the nodes on the road network graph has a list of 'nearest alarms' based on 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. We give a sketch of the phase one of this algorithm which performs the road network partitioning among alarm nodes in Fig. 6 [12].

The road network partitioning among alarm nodes proceeds in the following steps. First, given a total of n alarms, and a weighted directed graph G = (V,E) where V is the number of nodes and E the number of edges, we first determine the points of intersection of the alarm area with the roads on the road network graph and generate a tentative road network graph by adding these points of intersections as intermediate vertices to the road network graph. Then we need to partition the V nodes into n groups (one group for every alarm). The partitioning process uses three vectors: *distance*, *final* and *partition* and each is of length $|V|$. At each iteration, the i^{th} element in the distance vector, denoted as $distance_i$, represents the currently known shortest distance to the alarm stored in $partition_i$. The value in $final_i$ denotes whether the computations for node i, i.e., $distance_i$ and $partition_i$ have been finalized or not. If the node i with the least value of $distance_i$ has not been finalized yet ($final_i \neq 1$), then both the node i and the nodes which can directly lead up to the node i are chosen. Their *distance* and *partition* values are updated if the new path through node i is of shorter distance. After all nodes have been exhausted, the values in *partition* vector represent the network Voronoi diagram and it will be used to perform the alarm checking. Figure 5d shows the partitioning of the vertices of a road network graph. In the illustration, each alarm has a shape (color) associated with it and the vertices of the road network graph for which that particular alarm is the nearest (road network distance) are represented by the same shape (color).

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. For instance, $N \approx 35{,}000$ for the entire city of Atlanta and $N \approx 10{,}000$ for the District of Columbia [1]. Consider a dire case scenario where each node on the graph has two 'nearest' alarms (requiring 8 bytes of storage with the assumption that each of the alarms is represented as 32 bit integers), the total storage cost would be 280 and 80 KB for Atlanta and DC respectively, which can be easily accommodated in most of existing mobile devices. At each wakeup, only one of the two cases will be satisfied: either the mobile client is on a road (i.e., on the edge of the road network graph) or the mobile client is at the vertex on the road network graph. In the former case both the vertices

Algorithm 1: Road Network Partitioning amongst Alarm Nodes

Input:
$G(V, E)$ - Road Network Graph G with V vertices/nodes and E edges/roads
A - A set of nodes $A \subset V$ where each element (referred to as alarm node A_i) is enclosed by at least one alarm.
Output: Return a set *'partition'* of length $|V|$ representing the partitioning of each of the nodes in the road network G amongst the different alarm nodes A. Each element of this set $partition_i$ would contain a reference to the alarm node A_j to whose group the current node i belongs

```
/* Initialize the scratch space */
/* 'distance' is a vector of length |V| such that distance_i = 0 if ∃ Alarm Node A_j and
   V_i = A_j where 1 ≤ j ≤ |A| and 1 ≤ i ≤ |V|. At each iteration distance_i will represent
   the shortest distance from current node V_i to alarm node in partition_i. In other
   words only the Alarm Nodes are at a distance of 0 from their respective groups
   while the other nodes distances are unknown. */
/* 'final' is also a vector of length |V| such that final_i = 0 if the computational
   elements i.e. distance_i and partition_i have been finalized for the node V_i and 1
   otherwise. */
/* 'partition' nodes: At the beginning only the 'Alarm Nodes' are present in their
   respective groups with the status of the other nodes unknown (represented as ∞) */
1  for i from 1 to |V| do
2  |  if V_i = A_j for some j such that 1 ≤ j ≤ |A| then
3  |  |  distance_i ← 0
4  |  |  partition_i ← A_j
5  |  else
6  |  |  distance_i ← ∞
7  |  |  partition_i ← ∞
8  |  end
9  |  final_i = 0
10 end

   /* Repeat till there are no more non-finalized nodes and choosing the node with the
      least 'distance' value at each iteration. */
11 while ∃c such that 1 ≤ c ≤ |V| and final_c ≠ 1 and distance_c ≤ distance_j where 1 ≤ j ≤ |V| do
12 |  foreach Node N_i that can reach Node V_c without any intermediate hops do
   |  |  /* Is it possible to reduce distance_i if the path goes through V_c? */
13 |  |  if distance_i > distance_c + w then
   |  |  |  /* w is the weight of the edge from V_c to V_i */
14 |  |  |  distance_i ← distance_c + w
15 |  |  |  partition_i ← partition_c
16 |  |  end
17 |  end
18 |  final_c = 1
19 end

20 return partition
```

Fig. 6 Constructing the network Voronoi diagram

at the ends of the road are chosen while in the latter case the current vertex is chosen. Each of these vertices will have the list of nearest alarms associated with them and each of the alarms is checked for satisfaction upon wakeup. Even in this assumed dire case scenario, it would require only four (constant) number of alarm checks on each wakeup.

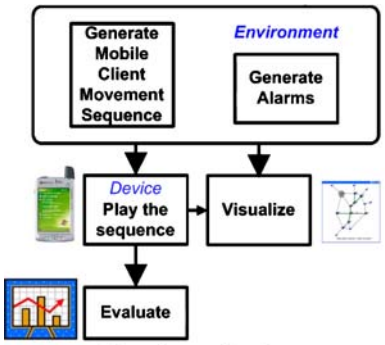
5 Experimental evaluation

We have described the four wakeup algorithms and the three alarm check algorithms. In this section we evaluate our middleware 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.

5.1 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

Fig. 7 Simulator design



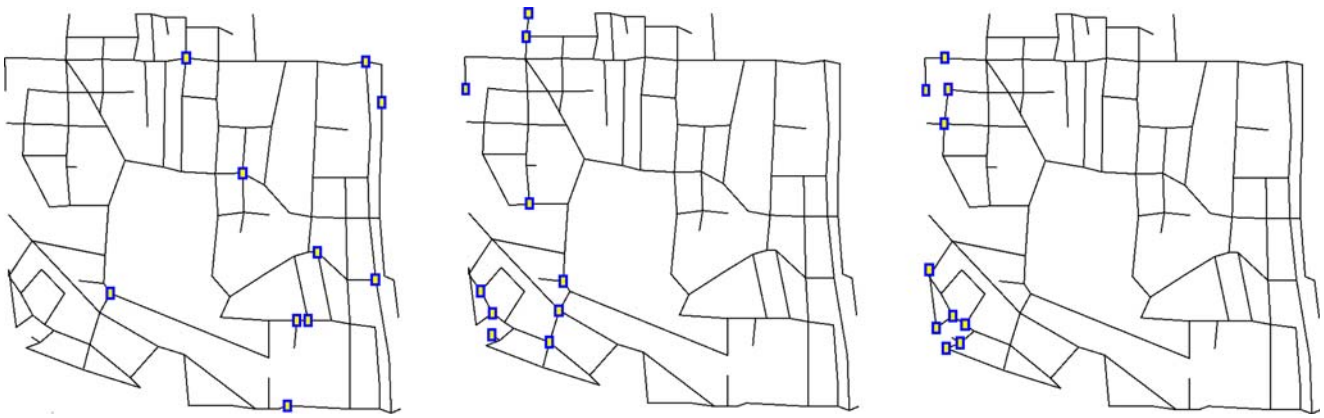
Simulator Design

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

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 Figs. 3 and 8. The *Evaluation* module comprises scripts to systematically vary environmental parameters listed in the simulator parameter table given in Fig. 7, which generate performance graphs presented in the rest of Section 5.

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. Figure 8 depicts the distribution of alarms using three different clustering factors. When the clustering factor is set to one, it represents the case where the alarms of a client is distributed over the entire map. When a clustering factor is set to two, it represents the scenario where the alarms of a mobile client are distributed over only one half of the map. When we set the clustering factor to be a higher value like ten, it represents the scenario where all the alarms are distributed over only $\frac{1}{10}^{th}$ of the map.



(a) CF: 1 (b) CF: 3 (c) CF: 10

Fig. 8 Clustering factor (CF)

5.2 Estimating energy consumed 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 [3] given in the energy parameter table of Fig. 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 actual alarm checks performed to the maximum checks that can be performed $N_{A_{MAX}}$ in the minimum wakeup duration 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.

$$E_t = T_w P_w + T_s P_s \tag{11}$$

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:

$$T_w = N_w \times T_m \tag{12}$$

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:

$$T_s = T_t - N_w T_m \tag{13}$$

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 checked 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 ideal state, and C_r denote the number of alarm checks required. The lower the *Alarm Check Ratio* is, the lesser the number of alarms need to be checked, and hence the higher the energy conservation will be. Thus we can compute the power consumed during wakeup, denoted as P_w, by the following formula:

$$P_w = C_r P_{100} + (1 - C_r) P_i \tag{14}$$

Where C_r can be defined in terms of the number of actual alarm checks (N_C), the maximum number of alarm checks that can be performed ($N_{A_{MAX}}$), and the number of wakeups (N_w):

$$C_r = \frac{N_C}{N_{A_{MAX}} * N_w} \tag{15}$$

By substituting Eqs. 12, 13, and 14 in Eq. 11, we obtain the *Total Energy consumed* as follows:

$$E_t = N_w T_m (C_r P_{100} + (1 - C_r) P_i) + (T_t - N_w T_m) P_s \tag{16}$$

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:

$$T_b = \frac{C_b \times T_t}{E_t} \tag{17}$$

5.3 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.

5.3.1 Overall energy consumption and battery lifetime

Figures 9, 10, 11 compare energy and battery life time for a mobile device by varying the number of alarms per map tile for different wakeup algorithms (Fig. 9), different alarm check algorithms (Fig. 10), and five alternative combinations of wakeup algorithm and alarm check algorithm (Fig. 11). We used the measurements for the Itsy Pocket Computer [3] given in the energy

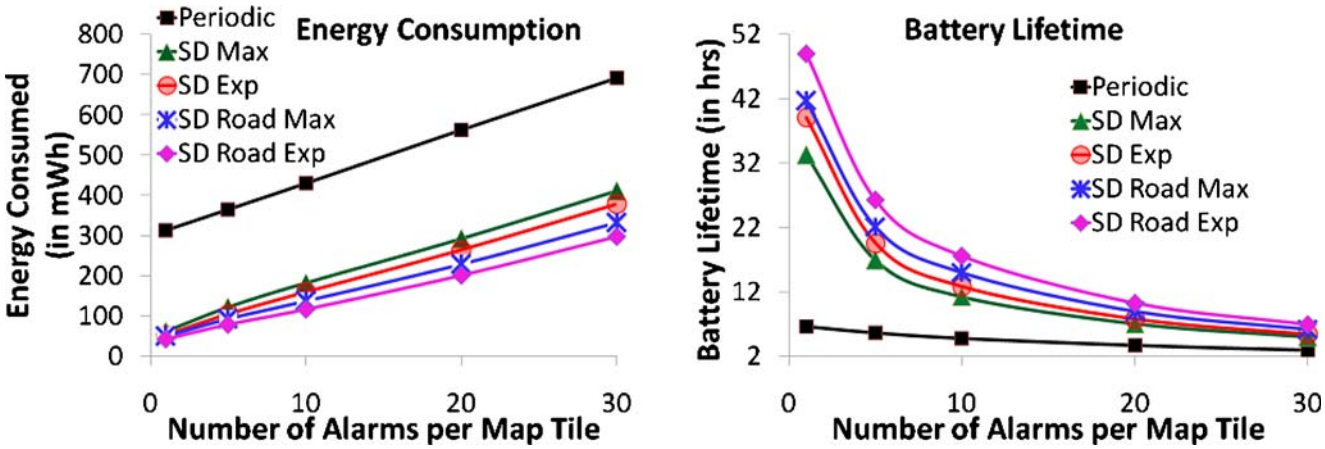


Fig. 9 Performance of WakeUp strategies

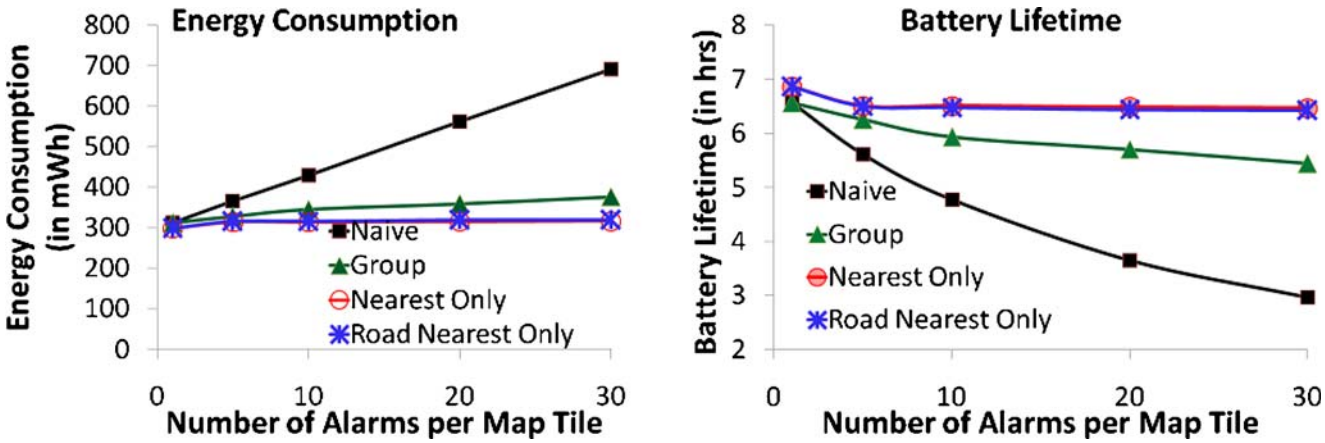


Fig. 10 Performance of check strategies

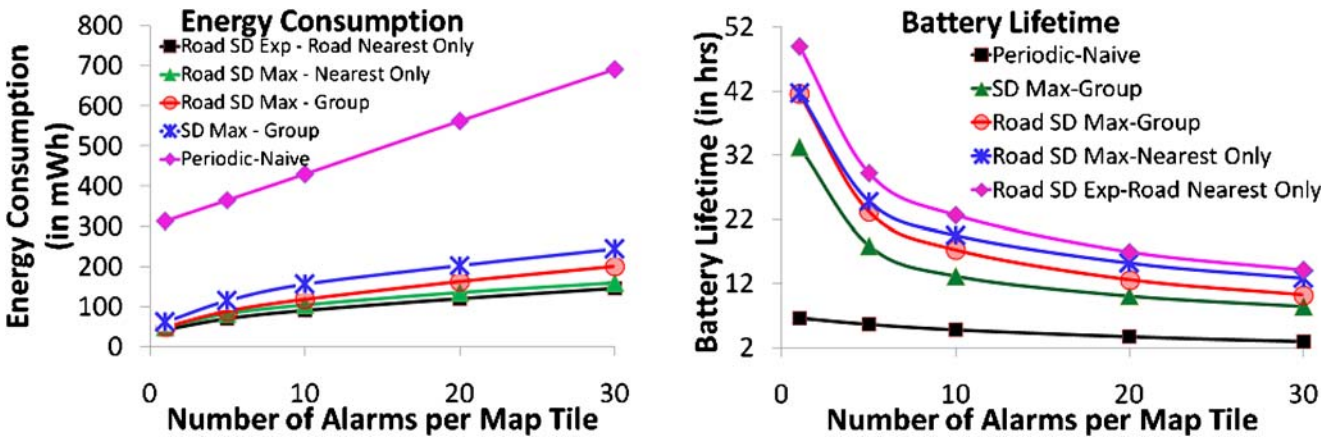


Fig. 11 Performance of WakeUp-Check strategy combinations

parameter table of Fig. 7 for these experiments. The results were plotted using the equations in Section 5.2. Apart from the values listed in the table in Fig. 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 h. 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 9 (left) compares 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 six 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 9 (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 always sufficient to reduce the energy consumption involved. For such scenarios, alarm check algorithms are necessary for energy conservation purpose.

Figure 10 (left) compares the energy consumption of the alarm check algorithms by fixing the wakeup Strategy (Periodic WakeUp Strategy is used in this case). We 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 10 (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. 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 11 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 Fig. 11 (right) is similar to Fig. 9 (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.

5.3.2 Effect of changes in number of alarms

Figure 12 (left) compares 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 speed* reduce the frequency of wakeups further. However, as shown in Fig. 14 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 12 (right) compares 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 [13] 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.

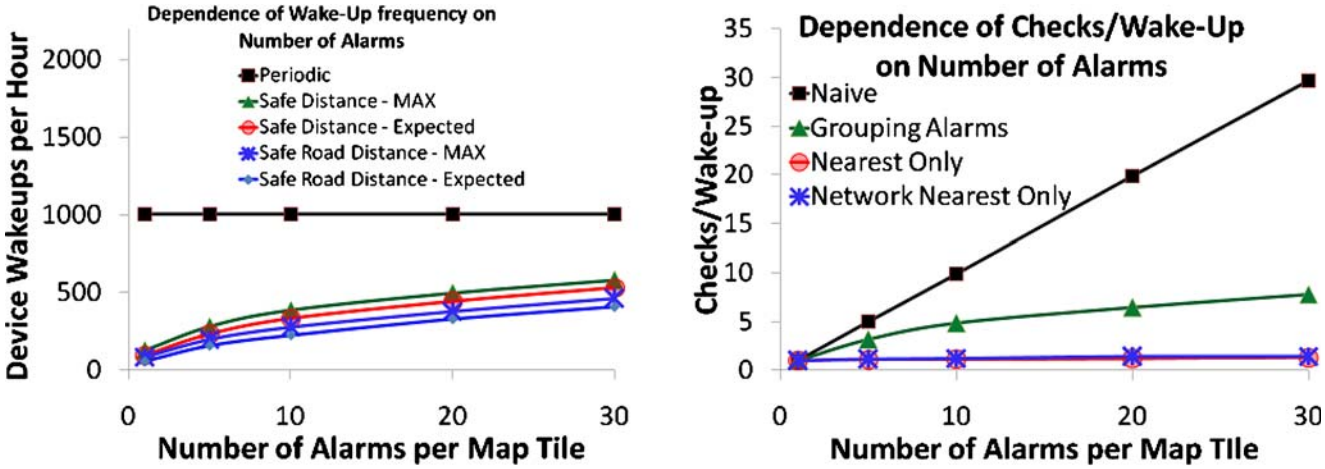


Fig. 12 Effect of changes in number of alarms

5.3.3 Effect of changes in alarm distribution

Figure 13 shows 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 location is higher, and the client has longer time to sleep.

Figure 13 (right) shows 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 xfaction.

5.3.4 Alarm delivery quality

Figure 14 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*

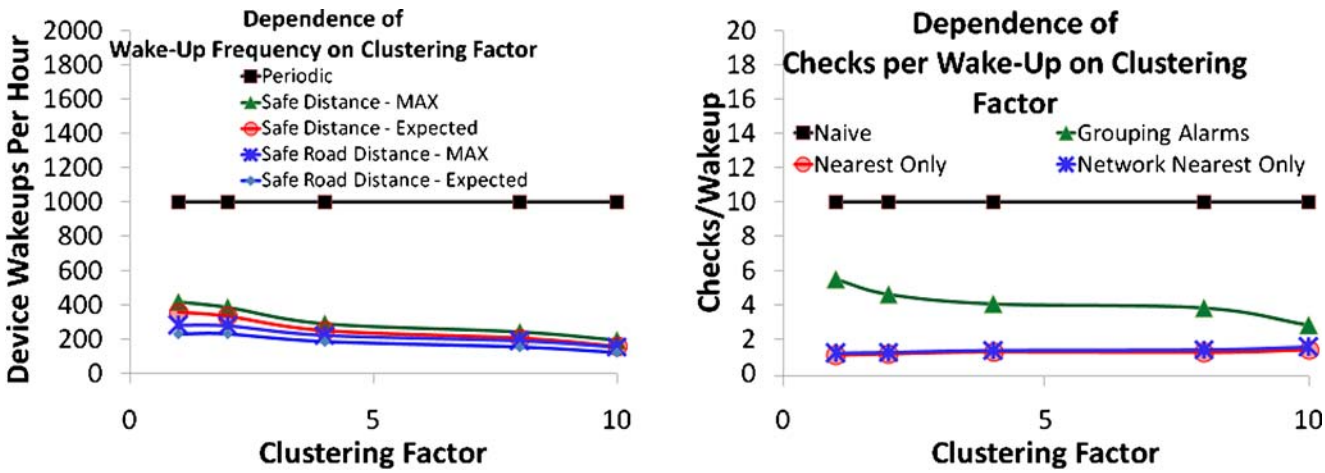


Fig. 13 Effect of clustering of alarms

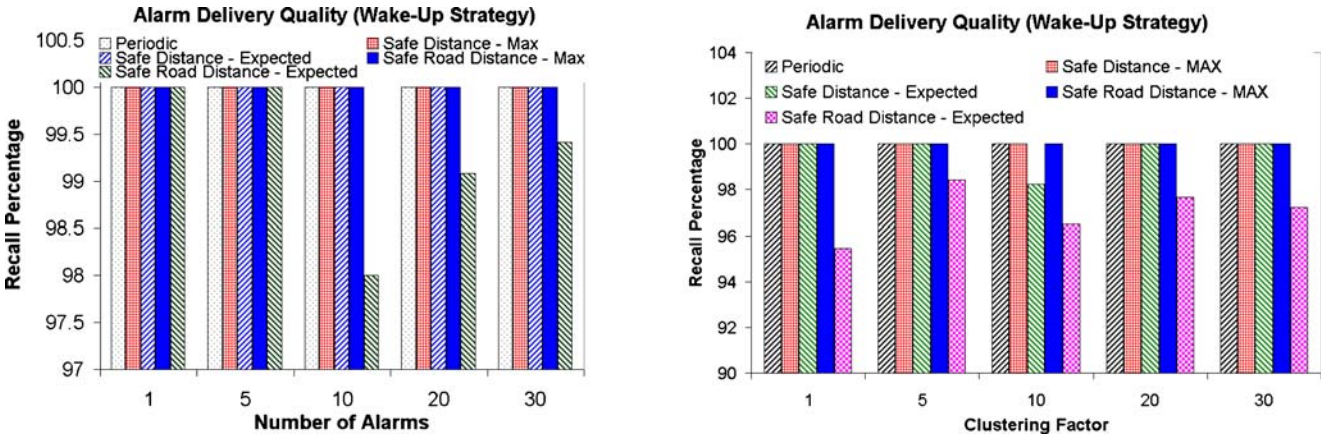


Fig. 14 Alarm delivery quality

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* approach 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 wakeups 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.

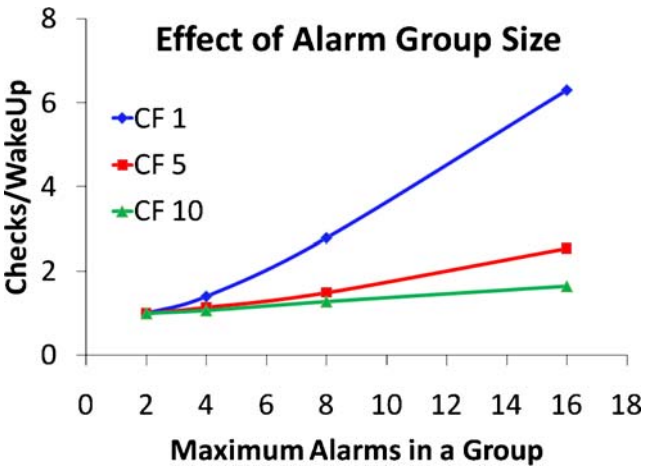


Fig. 15 Effect of alarm group size

5.3.5 Effect of alarm group size

Figure 15 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 [13] 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 the more worse the performance of alarm process will be. This is particularly the case when the alarms are scattered all around the map ($CF = 1$).

5.3.6 Storage requirements for nearest alarm only check algorithm

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 (Fig. 16 (left)), while at the same time the number of checks required at each wakeup is reduced at a very high rate (Fig. 16 (right)).

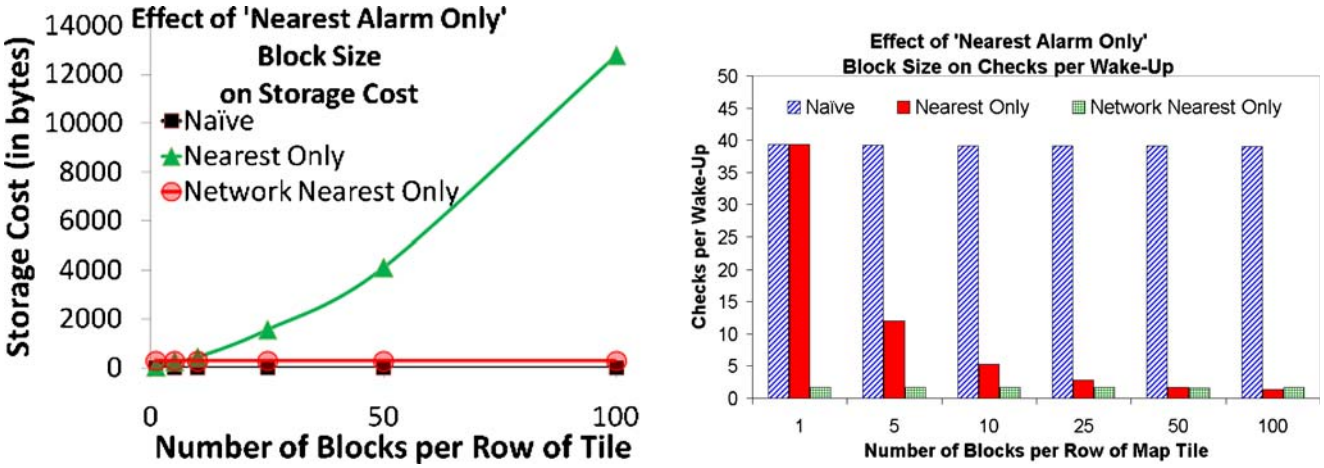


Fig. 16 Storage requirements

6 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 [4, 6, 15, 16, 20, 21]. These papers primarily concentrate on usability of such location reminder systems from the human–computer interaction point of view. For instance, [14] 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 much works like [9, 10, 18], which describe general strategies to minimize energy consumptions on mobile devices, such as *Dynamic voltage scaling*, and to adapt mobile applications to consume lesser energy. There has also been research in the area of continuous spatial queries like in [17, 19, 22]. Our work in this paper distinguishes itself primarily along two dimensions—(a) addressing the special class of one time queries i.e. detecting the spatial validity of the alarm and triggering the action if conditions are satisfied as opposed to continuously keeping track of the objects that satisfy the nearest-neighbor or range query and (b) by addressing the energy consumption aspects as opposed to addressing only scalability or performance efficiency.

We have presented a middleware architecture for energy efficient processing of 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. We evaluate the scalability and energy-efficiency of our wakeup and alarm check approach using a road network simulator. The experiments show 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 each wakeup.

Acknowledgements This work is partially supported by grants from NSF CISE under CNS, NetSE, and CyberTrust, a grant from AFOSR, an IBM SUR grant, and an IBM faculty award, and an Intel research council grant. We would also like to thank Ramesh Chandran, Kipp Jones, Calton Pu, and the mobile computing group within the DiSL lab for the valuable discussions and help with the experiments.

References

1. US Census Bureau, Tiger, tiger/line and tiger-related products. Available at http://www.census.gov/geo/www/tiger/
2. Aurenhammer F (1991) Voronoi diagrams a survey of a fundamental geometric data structure. ACM Comput Surv 23(3):345–405
3. Bartlett JF, Brakmo LS, Farkas KI, Hamburgen WR, Mann T, Viredaz MA, Waldspurger CA, Wallach DA (2000) The itsy pocket computer. Technical Report 2000/6, COMPAQ Western Research Laboratory

4. Bazinette V, Cohen NH, Ebling MR, Hunt GDH, Lei H, Purakayastha A, Stewart G, Wong L, Yeh DL (2001) An intelligent notification system. IBM research report
5. Cherkassky BV, Goldberg AV, Radzik T (1993) Shortest paths algorithms: theory and experimental evaluation. Math Program 73:129–174
6. Dey AK, Abowd GD (2000) Cybreminder: a context-aware system for supporting reminders. In: Handheld and ubiquitous computing: second international symposium, HUC 2000, Bristol, UK proceedings
7. Dijkstra EW (1959) A note on two problems in connection with graphs. Numerische Math 1:269–271
8. Erwig M (2000) The graph voronoi diagram with applications. In: Networks, pp 156–163
9. Flautner K, Mudge T (2002) Vertigo: automatic performance-setting for linux. SIGOPS Oper Syst Rev 36(SI):105–116
10. Flinn J, Satyanarayanan M (1999) Energy-aware adaptation for mobile applications. In: SOSP '99: proceedings of the seventeenth ACM symposium on operating systems principles. ACM Press, New York, NY, USA, pp 48–63
11. Fortune S (1997) Voronoi diagrams and delaunay triangulations. In: Goodman JE and OŠRourke J (eds) Handbook of discrete and computational geometry. CRC, Boca Raton, pp 377–388
12. Graf M, Winter S (2003) Netzwerk-voronoi-diagramme. Österr Z Vermess Geoinf 91:166–174
13. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: SIGMOD '84: proceedings of the 1984 ACM SIGMOD international conference on management of data. ACM, New York, pp 47–57
14. Heidemann J, Shah D (2000) Location-aware scheduling with minimal infrastructure. In: USENIX conference proceedings. USC/Information Sciences Institute, USENIX, San Diego, pp 131–138
15. Kim SW, Kim MC, Park SH, Jin YK, Choi WS (2004) Gate reminder: a design case of a smart reminder. In: DIS '04: Proceedings of the 2004 conference on designing interactive systems. ACM, New York, pp 81–90
16. Ludford PJ, Frankowski D, Reily K, Wilms K, Terveen L (2006) Because i carry my cell phone anyway: functional location-based reminder applications. In: CHI '06: proceedings of the SIGCHI conference on human factors i computing systems. ACM, New York, pp 889–898
17. Mokbel MF, Xiong X, Aref WG (2004) Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In: In SIGMOD, pp 623–634
18. Mudge T (2001) Power: a first-class architectural design constraint. In: Computer. Michigan Univ., Ann Arbor, pp 52–58
19. Prabhakar S, Xia Y, Kalashnikov DV, Aref WG, Hambrusch SE (2002) Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects. IEEE Trans Comput 51(10):1124–1140
20. Sadeh NM, Gandon FL, Kwon OB (2005) Ambient intelligence: the mycampus experience. In: CMU-ISRI-05-123
21. Sohn T, Li KA, Lee G, Smith I, Scott J, Griswold WG (2005) Place-its: a study of location-based reminders on mobile phones. In: UbiComp 2005: ubiquitous computing
22. Zhang J, Zhu M, Papadias D, Tao Y, Lee DL (2003) Location-based spatial queries. In: In SIGMOD, pp 443–454