Energy-Efficient Monitoring of Mobile Objects with Uncertainty-Aware Tolerances

Tobias Farrell
Institute of Parallel and Distributed Systems, Universität Stuttgart, Germany

Reynold Cheng
Department of Computing, Hong Kong Polytechnic University, Hong Kong

Kurt Rothermel
Institute of Parallel and Distributed Systems, Universität Stuttgart, Germany

Abstract

In location-based services, continuous queries are often employed to monitor the locations of mobile objects that are determined by sensing devices like GPS receivers. Due to limited battery resources, it is important for these objects to acquire and report location data only if necessary. We study how these energy-consuming operations can be reduced with a controlled impact on query accuracy of continuous range queries (CRQs). Specifically, we develop uncertainty-aware tolerances*, which are user-defined error bounds that provide correctness guarantees, with consideration of different sources of data uncertainty: sensing uncertainty, sampling uncertainty, and communication delay. Novel algorithms are developed to control carefully when an object should acquire and update a location, while satisfying these tolerances. Extensive simulations validate the effectiveness of our methods.*

1. Introduction

Due to the rapid development of low-cost location-sensing devices, like the Global Positioning System (GPS), and wireless networking technologies, location-based services have attracted tremendous research interest lately [10],[12],[26]. In particular, long-standing, *continuous* queries are used to monitor various activities of mobile objects for an extensive period of time. A wide range of applications has been identified, including intrusion detection over security-sensitive regions, mobile advertisements for customers nearby, and traffic monitoring.

In such systems, each mobile object is equipped with a *sensing* device (e.g., a GPS receiver) to acquire location data, and a *wireless communication* interface (e.g., a GSM/GPRS transceiver) to report data to a location server. The sensing and reporting operations constitute the major fraction of energy consumed in a mobile object [23]. Since the battery resources of many mobile objects (like cellular phones and PDAs) are precious, it is important to minimize the usage of these operations, so that the lifetime can be maximized. This issue is particularly critical for continuous queries, which require location data to be constantly sensed and reported to the server for further processing.

In order to reduce costs of continuous query processing, the idea of *object-side processing* has been utilized in various projects [3],[9],[17],[22]. These works are based on the idea that a mobile object has some processing capabilities to decide by itself whether to report a data item to the location server. Specifically, upon receiving a query request, the server sends some query information to each mobile object. The object then evaluates part of the query locally and transmits the location to the server only if the query result is affected. As an example, consider a *Continuous Range Query (CRQ)*, which returns the identities of mobile objects located inside some query region with boundary R. If this information is propagated to all mobile objects, then an object only needs to send a position update to the server if it crosses R. Since fewer items are reported to the server, the number of messages, as well as the energy costs incurred in reporting operations, can be reduced significantly.

Although object-side processing is a promising way of reducing energy use, there are two open challenges. The first is handling of *data uncertainty*, inherently associated with a location item. For example, a location value obtained by some GPS receiver is only correct within a few metres (called *sensing uncertainty*) [27]. Another source of uncertainty, known as *sampling uncertainty*, is produced when the locations are only sensed at discrete time instants. Then, the positions between adjacent samples are not precisely known [21]. Moreover, *communication delay* causes the location data to be received some time after sensing. Data uncertainty affects the accuracy of query results. In a CRQ, for instance, an object may temporarily leave the monitored region unnoticed,

11th International Database Engineering and Applications Symposium (IDEAS 2007)
0-7695-2947-X/07 $25.00 © 2007 IEEE

since it acquires a location at discrete time instants only. More importantly, the query result is usually updated late – at some time after an object crossed *R*. Thus, the query result cannot correctly reflect the real world at all times.

The second challenge is that energy consumption issues for *sensing* the location of a mobile object have not been well studied. While most previous works focus on communication costs [1],[3],[4],[19],[20],[22], very often the energy required for sensing at a mobile device cannot be ignored [23]. Even in low power mode, a common GPS receiver consumes no less than 75 mJoules for each position acquired [18]. The same amount of energy is required by GPRS to transmit 120 bytes of data [9]. Consequently, the design of an energy-efficient query protocol should consider the energy consumed by position sensing, too. In Section 3 we show how this factor of energy consumption can be reduced by carefully controlling the sensing rate. For example, the positioning sensor may remain in low-power mode for a longer time, if the object is located far away from *R*. However, this reveals an important trade-off between the frequency of sensing and accuracy of query results. A mobile object near the query boundary has to sense its position with higher frequency in order to provide better accuracy. As a result, object-side processing of CRQ requires a significant amount of energy to provide the best possible query result – while absolute query correctness cannot be guaranteed due to different sources of data uncertainty.

In this paper, we propose to overcome these two shortcomings by *relaxing* the query accuracy requirements. This is motivated by the observation that many location-aware applications do not require the highest degree of query accuracy. Instead, they can relax the correctness requirements by specifying a maximum error bound in the query results. For example, consider a CRQ used for distributing warning messages within a spatial region. It is acceptable that some users receive the notification early (before entering the queried region). In contrast, for distributing location-based advertisements to users located inside a supermarket, some users could receive it late (after entering the store). We introduce the notion of *uncertainty-aware tolerances* for CRQ, which defines the maximal acceptable error along with the query. The allowed tolerances are then guaranteed to be met in consideration of all sources of uncertainty. Furthermore, it can be utilized to reduce the energy consumption of mobile objects. We present efficient algorithms that satisfy the tolerance constraints and also consider energy usage.

The basic idea of *error-tolerant query processing* has recently been exploited by various researchers [19],[20],[4]. Based on the trade-off between the frequency of reporting operations and query correctness, they have developed intelligent algorithms to achieve lower communication costs. However, reported sensor values were assumed to be *always* correct. In contrast, we propose a new notion of query tolerances that consider different sources of data uncertainty. Moreover, previous solutions have not considered the energy costs of *sensing* new data. As we will show, there is an important trade-off between the amount of energy spent on either sensing or reporting operations. In our paper this trade-off is controlled carefully in order to reduce the overall energy consumption.

In summary, our contributions are:

- Propose uncertainty-aware tolerances semantics for continuous range queries (CRQ);
- Show that uncertainty-aware tolerances provide correctness guarantees under three major sources of uncertainty: sensing uncertainty, sampling uncertainty, and communication delay;
- Develop efficient algorithms for processing CRQ, that satisfy uncertainty-aware tolerances and reduce the total energy consumed by sensing and reporting operations; and
- Verify the effectiveness of our approaches by extensive simulation using realistic mobility traces.

The rest of this paper is organized as follows. In Section 2 we describe our system model and detail the query model studied in this paper. Section 3 then analyzes a preliminary solution to disclose existing shortcomings of processing CRQ. Subsequently, we present how to overcome these shortcomings by introducing uncertainty-aware tolerances in Section 4. Section 5 presents our experimental results and Section 6 discusses related work. Finally, we conclude the paper in Section 7.

2. Background

We now describe the system architecture, properties of a mobile object and all basic assumptions. Then, we present the underlying principle of object-side query processing that is studied in this paper.

2.1 System Model

Our system model consists of mobile objects (MOs) and a location manager (LM). The LM processes continuous queries on behalf of location-aware applications (LAs). We do not make any assumptions on the internal organization of the LM. It might comprise multiple LM nodes, to which the MOs are mapped (dynamically) [13]. Each MO communicates with a single LM node over a wireless network, such as GPRS, UMTS or WiFi meshes.

An MO is a mobile device (e.g., cell phone, PDA) equipped with a processor, a wireless network interface and a positioning sensor to detect its geographic position. Each MO is identified by a globally unique identifier O_i, where i=1,...,n and n is the total number of objects monitored by the system. To supply the LM with current position information, an MO has to perform three different operations: processing, communication and position sensing. We focus on the last two operations because they dominate energy consumption [8],[23].

(1) Communication*:* An MO is responsible to send update messages to the LM according to some query protocol. As all update messages will be similar in size, we assume this transmission requires a constant amount of energy W_U per message. To cope with unbounded communication delays, we assume a statistical upper bound of c_{max} for the end-to-end delay between an MO and the LM. This can be determined empirically based on the networking environment and holds with high probability [19]. Occasionally, messages may be delayed by more than c_{max} in practise. This will result in temporary violations of precision guarantees – an unavoidable effect in any distributed environment with unbounded delays.

Note that we aim at satisfying the tolerances for data stored on the LM in the following. If an LA runs on a different node, the transfer of query results causes an extra delay, which depends on the characteristics of the communication channel between LA and LM. In that case, only c_{max} must be adapted accordingly, to ensure that the tolerance constraints still hold when the LA receives the data. The algorithms proposed in this paper are not affected by that.

(2) Sensing*:* We adopt a generic model derived from GPS technologies [16], which is applicable to a broad class of positioning sensors. Specifically, position sensing is not performed continuously to conserve precious energy. Instead, the positioning sensor determines its current location by performing a *position fix*. Each position fix is explicitly invoked by the processor and requires some amount of time T_{sense} before the position is obtained. For example, GPS needs about 0.5 s for pseudo-range measurements of satellite signals and computing a valid position [16]. Hence, the maximum sampling rate of a positioning sensor is $1/T_{sense}$. Each position fix also requires a constant amount of energy W_S. In between two fixes the positioning sensor can operate in a low-power *sleep* mode to conserve energy.

Note that we do not consider any background energy that is not influenced by these two operations. Its consumption is independent of the reporting protocol. For instance, a GPS receiver might still wake up *periodically* to keep a lock on the satellite signals.

The real position of an MO at any time t is denoted $loc(O_i)$. However, sensing uncertainty generally causes the location data acquired by a positioning sensor to deviate from the real position. We assume that there is a maximum deviation S_{acc} from $loc(t)$ [27]. Furthermore, we assume that each MO has knowledge about its maximal velocity, denoted by v_{max}. This is a common assumption for tracking mobile objects and determining reasonable values has been discussed elsewhere, e.g., [21].

2.2 Query Model

In this paper we focus on *Continuous Range Queries* (*CRQ*). Given a closed region with boundary R, this query returns the identities of all mobile objects located inside R. In contrast to one-time queries, a CRQ resides in the system and is continuously evaluated for an extensive period of time. This type of query can be used to monitor objects moving into and out of a spatial region (e.g., if any child leaves a playground), which is an important building block of many location-aware systems [3],[9],[22].

Location-aware applications can *register* a CRQ at the LM. Then, the query remains *active* until it *deregistered* again. Since the locations of MOs change frequently, the result of an active query has to be refreshed timely. To do this, the LM sends a *result message* to the LA after registration and whenever the result changes. The time to refresh a result depends on the objects' movements. Specifically, the result of a CRQ must be updated whenever an MO enters or leaves the query region. That is, whenever an object crosses R, the *query boundary*.

For efficient query processing, we employ the concept of *object-side processing* [3],[9],[17],[22]. That is, the LM collaborates with the MOs to optimize query processing. It conveys query information to them over a wireless network, where it is then utilized to evaluate part of the query locally over each location item acquired. Specifically, the details of a query are first propagated to all relevant MOs in an *init message*. All MOs located inside the queried region will then respond with an *update messag*e. Subsequently, each MO monitors its own location and sends a new update whenever it crosses the query boundary R. In the update message, it reports its current relation to R (*inside or outside*). For long-running continuous queries this approach minimizes the energy spent on communication because the costs for receiving the query at the beginning are easily amortized by saving update messages throughout the query's lifetime.

In the following we focus on the local query evaluation performed by each MO (i.e., the object-side processing of CRQ). In particular, we investigate in detail when

an MO must perform a position fix or generate an update message in order to meet the query requirements. Minimizing these operations is essential to reduce energy consumption.

3. Problem Analysis

In this section, we first discuss a straightforward solution to reduce the energy spent on sensing for object-side processing of CRQ. This preliminary approach will help to identify major shortcomings and motivate our solutions.

Whenever a CRQ is active, all MOs must locally monitor their position, to detect crossing the query boundary in time. But due to high energy consumption, position sensing is usually not performed continuously. Instead, an MO can use a technique called *selective sensing* to conserve more energy: After each position fix the MO computes the time it can suspend sensing without affecting the query result. This is the minimum amount of time required to reach the query boundary based on the MO's maximal velocity. If the update threshold is not yet reached after that time, the next fix can be scheduled based on the remaining distance to the query boundary.

The resulting algorithm for object-side processing is depicted in Figure 1. For a query's lifetime, the following steps are repeatedly executed: First, a new position is obtained from the positioning sensor. Subsequently, the function `mustUpdate` checks if the object crossed the query boundary since the last position fix. For that purpose, the MO's previous state is recorded using a boolean variable `isInside` (line 1). Whenever this value differs from the object's new state (line 16) the query result must be updated. Thus, a new update message is sent to the LM. Finally, the minimum amount of time to reach the query boundary is computed based on the object's maximum velocity (line 17 and 18). Since the MO cannot affect the query result during this period of time the next position fix can be deferred accordingly (line 14). Waiting any longer might, however, violate the query condition of CRQ, as the future velocity is not yet known.

```
Main:
(1)   isInside := false;
(2)
(3)   while (query is active) do {
(4)        /* acquire new position */
(5)        newPos := readSensor();
(6)
(7)        /* check update */
(8)        if (mustUpdate()) {
(9)             isInside := NOT isInside;
(10)            sendUpdate(Oi, isInside);
(11)       }
(12)       /* low-power mode */
(13)       T_wait := T_max() - T_sense;
(14)       if (T_wait > 0) { sleep(T_wait) };
(15)  }

mustUpdate:
(16) return ((newPos ∈ R) != isInside);

T_max:
(17) d := dist(R, newPos);
(18) return d / v_max;
```

Figure 1. Object-side processing of CRQ without tolerance.

As a result, the algorithm in Figure 1 generates a new update message as soon as crossing the query boundary can be discovered. While doing so, it defers each position fix as long as possible without compromising timeliness of detection. Although simple, this algorithm exhibits a number of shortcomings. First, we observe that the algorithm cannot meet the defined query semantics precisely. Ideally, the LM should always update the query result exactly at the time an MO crosses the query boundary. However, this point in time might be missed due to the limitations of current sensor technologies and communication delay. In fact, we can quantify three different sources of uncertainty for location data:

(1) Sensing Uncertainty*:* Since the MO's future movement is not known in advance, the best time to send an update is when the sensed position is located just beyond the query boundary. Taking limited sensing accuracy into account, however, the acquired position can deviate from the MO's real position by a maximal distance of S_{acc}. That is, at the time an update is generated, the MO could still be approaching the query boundary. If it changed its direction of movement right after the position fix, the algorithm would generate an update even though the MO has never actually crossed the boundary. In the other extreme case, the MO might be located at a distance of S_{acc} beyond the boundary at that time. As a consequence, the algorithm then generates an update late – at some time after the MO affected the query result in reality. The effect of sensing uncertainty is also illustrated in Figure 2. At the time an update is generated, the MO might actually be located anywhere inside the shaded region around its assumed position p_1.

(2) Sampling Uncertainty*:* Additionally, a new position is not available at all times. This is because each position fix requires some time of T_{sense}. That is, after performing a position fix, the next position will not be available until after T_{sense}. Consider a sensed position that is very close to the query boundary, but does not trigger an update yet. Although the MO might overstep the query boundary right after, it can then travel a distance of $T_{sense} \cdot v_{max}$ before the next position fix completes (depicted by position p_2 in Figure 2). In the worst case, sensing uncertainty adds into the same direction and the algorithm in Figure 1 thus generates an update message at even greater distance to R.

0-7695-2947-X/07 $25.00

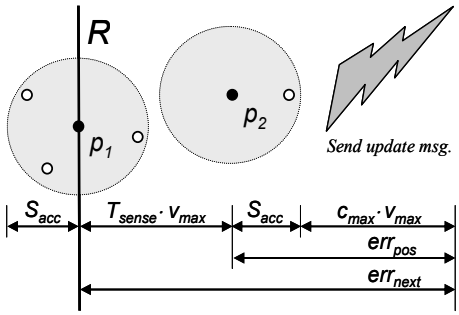

Figure 2. Error in sending an update with CRQ.

(3) Communication Delay in the network furthermore causes update messages to arrive at the LM some time after sensing. In the worst case, the MO moves away from the reported position at maximal velocity while the message is being transmitted. Recall that we assume a statistical upper bound of c_{max} for the end-to-end delay. That is, at the time the LM receives the information, the MO's real position is only known within the following distance from the acquired position (see Figure 2):

$$err_{pos} := S_{acc} + c_{max} \cdot v_{max} \quad (1)$$

By considering all three sources of uncertainty, the MO can have a maximum distance err_{next} from R at the time the query result is updated at the LM (see Figure 2), where

$$err_{next} := S_{acc} + (T_{sense} + c_{max}) \cdot v_{max} \quad (2)$$

As a consequence, the server-side result of a non-tolerant CRQ is always outdated. At any time, it suffers an error of ± err_{next} because the updates of both objects entering and leaving the query region arrive late. Note that this is not only a shortcoming of the presented algorithm but a technical limitation caused by limited capabilities of sensor systems and communication delay.

Next, let us examine the energy spent on position sensing. The algorithm in Figure 1 has been designed for minimizing the number of position fixes, without compromising query accuracy. In between two fixes, the positioning sensor remains in low-power sleep mode until the MO might reach R (line 14). A larger sleeping period would affect the provided accuracy, because the MO might not detect that it has crossed the query boundary on time. Consequently, the maximal error in the query result would further increase.

Figure 1 also reveals that the frequency of position fixes increases whenever the MO approaches the query boundary. This is because the sleeping time between two position fixes depends on the remaining distance to R (see line 17). The smaller this distance, the less time the sensor can remain in low-power mode. Once the object moves close to R, position fixes have to be performed at the highest frequency in order to generate the next update message in time. Furthermore, a short sleeping time does not allow the MO to cover a large distance before the next position fix is performed. Thus, the MO is still located relatively close to the boundary after that fix. As a consequence, the power consumption may increase substantially while the MO approaches the query boundary.

This reveals a critical problem of object-side query processing: although the defined semantics cannot be met completely, an MO has to invest a lot of energy in performing frequent position sensing in order to provide the best possible accuracy. In the next section we discuss how to overcome these two shortcomings by relaxing the query semantics.

4. Distance-tolerant CRQ

The major problem of evaluating a non-tolerant CRQ is that the MO cannot deduce precisely when it reaches the query boundary. As discussed, this requires a lot of energy for position sensing, and yet uncertainty is still introduced. Let us study how these shortcomings can be overcome by introducing *uncertainty-aware tolerances*.

4.1 Definition and Semantics

Our main idea is to define the maximum allowed error related to updates along with each query. This allows applications to specify their requirements more precisely. In exchange the allowed tolerances are guaranteed to be met in consideration of all sources of uncertainty, and valuable energy of MOs can also be conserved. To achieve these goals, the boundary of R is "blurred". That is, we introduce two distinct boundaries R_1 and R_2 such that R is sandwiched between them. The query result can then be updated *while* an MO crosses the region between R_1 and R_2, which we call the *tolerance region*. Let us first look at the following definition of distance-tolerant CRQ:

Definition 1: Distance-tolerant CRQ (d-CRQ)
Given two closed regions with boundaries R_1, R_2 ($R_1 \in R_2$, $dist(R_1,R_2) > 2 \cdot err_{next}$), a d-CRQ returns a set of ids that contains all MOs located in R_1 but no MO located outside R_2. That is, it returns the set $S \cup T$, with $S := \{O_i \mid loc(O_i) \in R_1\}$ and $T \subseteq \{O_i \mid loc(O_i) \in R_2\}$ ($1 \le i \le n$).

In this definition, the query result must contain all objects that are located inside of R_1 (the set S). However, objects located inside the tolerance region might also be contained in the result set. These objects belong to the set T, which is any subset of mobile objects located inside R_2 – including those that are located out-

0-7695-2947-X/07 $25.00 © 2007 IEEE

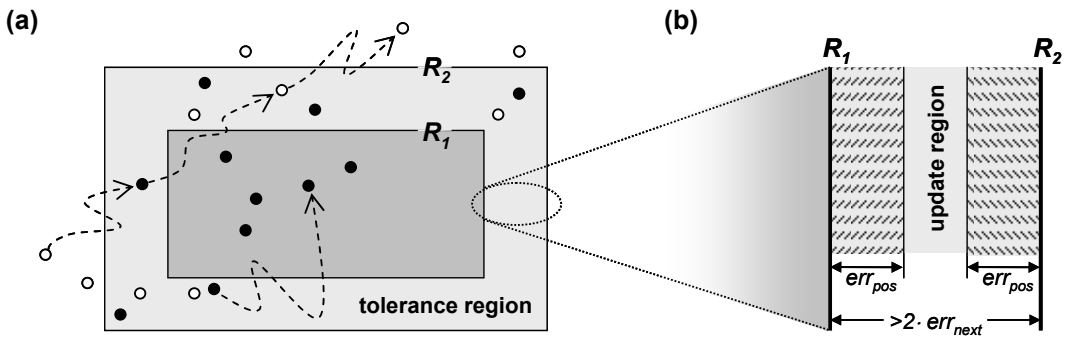

Figure 3. Definition of distance-tolerant CRQ with (a) tolerance region and (b) update region.

side R_1. However, no MO located outside R_2 is contained in the query result. This is illustrated in Figure 3a, where the tolerance region is lightly shaded. Black points depict MOs included in the result set, and the MOs not included are represented as white points.

Furthermore, note that Definition 1 is independent of the original query region R. An application can choose how to derive R_1 and R_2 thereof, depending on its requirements. For example, it can query all MOs located inside R_1 while accepting some *false positives*. The set of MOs falsely included in the result set is then bounded to MOs still located inside the larger boundary R_2. Alternatively, all MOs located inside R_2 can be queried if some *false negatives* are acceptable. Then, the smaller boundary R_1 in turn limits the set of objects missing in the query result. Balancing between these two extremes is also possible.

More importantly, Definition 1 offers well-defined bounds that can be detected in practice. The tolerance constraints are guaranteed with respect to the objects' real positions at the time a result update is received. That is, the query result contains the specified objects at all times – in spite of sampling uncertainty, sensing uncertainty and communication delay. To guarantee this, an update must be *received* while the MO is located inside the tolerance region. Consider an object initially located outside the larger region R_2 (Figure 3). The object must send an update after entering R_2, so that the update is received before it enters R_1. This ensures that the MO has already been included in the result set by the time it enters R_1. After sending an update message, the MO is free to move anywhere inside R_2 without violating the query constraint. However, it must ensure that another update is received by the LM before it leaves R_2 again. At that time it is in line with the query definition to remove the object from the result set, which must be accomplished before it actually leaves R_2.

To ensure an update can always be received in time, with consideration of uncertainty, Definition 1 requires the shortest distance between both boundaries $dist(R_1,R_2)$ to be no less than $2 \cdot err_{next}$ (Equation 2). Let us understand this requirement by examining the object-side processing of d-CRQ.

4.2 Object-side processing of d-CRQ

For d-CRQ, the MO initially receives both query boundaries R_1 and R_2 within the query init message. In the following, we describe how to generate position fixes and updates that conform to the tolerance constraints. Let us start with the following definition:

Definition 2: Update region is the region where an update message can be generated without violating the tolerance constraints.

As shown in Figure 3b the update region is located inside the tolerance region with a distance of err_{pos} (Equation 1) to each query boundary. Recall that err_{pos} is the maximal error between a sensed position and the MO's position at the time the update is received, when taking sensing uncertainty and communication delay into account. Thus, only updates generated at a larger distance to both boundaries are definitely received within the defined tolerance bounds.

To generate update messages accordingly, the main algorithm described in the previous section (Figure 1) is extended as shown in Figure 4[1]. Consider an MO located outside R_2. As discussed before, the object must make sure that it sends an update before it enters R_1. Accordingly, the function `T_max` computes the shortest time to reach R_1 from the current position of the object (line 21). This value must be reduced by err_{pos} in order to determine the maximum distance before an update must be sent. A later update could not be guaranteed to arrive before the object enters R_1 in reality. Subsequently, the shortest time to cross that

[1] Let `signedDist(R,p)` be a function that returns the shortest distance to enter the region defined by R from the position p. If p is already located inside R the shortest distance to its boundary will be returned with a negative leading sign instead.

0-7695-2947-X/07 $25.00 © 2007 IEEE

distance is computed based on the object's maximal velocity (line 23). This results in the longest time the next position fix can be deferred without violating the tolerance constraints. However, an update must be sent immediately if the subsequent position fix is not known to complete within the remaining time ($T_{max} < T_{sense}$, line 2). On that account, the defined tolerance of the query allows updating earlier – at any time the object is located inside the update region.

To assure this constraint, the function `T_min` calculates the time until an update might be sent at the earliest. Taking sensing uncertainty into account, the query definition would actually allow sending an update as soon as the sensed position is located inside of R_2 with a distance larger than S_{acc}. However, we also have to consider the object's situation after sending an update. Then, it has to monitor when it leaves the region R_2 again. Once the object is considered part of the query result, it must ensure another update message is *received* before it leaves R_2 again. Thus, we must ensure that the MO has sufficient time to execute at least one more position fix before this constraint is violated. Consequently, the first update message must be deferred until the distance to R_2 exceeds err_{next} (line 7). This observation also requires the minimum distance between both regions to be guaranteed. If an object was located at a distance any smaller than err_{next} to both regions at the same time, it would be unable to fulfil the query semantics. For that reason Definition 1 required $dist(R_1,R_2)$ to exceed $2 \cdot err_{next}$ at first hand.

Finally, if none of the conditions was fulfilled yet, the object must be located somewhere in the update region with a distance larger that err_{next} to both query boundaries (line 12). In this case no update message has to be sent yet, but it is already allowed. The function `updatePolicy` (line 13) then decides whether the MO should send an update immediately or wait for the next position fix to complete. Regarding the energy consumption, this is an important part of the algorithm. Here, the energy costs of sensing and reporting must be balanced to reduce the total amount of energy consumed. We discuss suitable strategies to balance between these two factors of energy consumption in the next subsection.

```
mustUpdate:
(1)  /* check upper bound */
(2)  if (T_max() < T_sense) {
(3)        return true;
(4)  } // else:
(5)
(6)  /* check lower bound */
(7)  if (T_min() > 0) {
(8)        // must not update yet
(9)        return false;
(10) } // else:
(11)
(12) /* between the bounds */
(13)       return updatePolicy();
(14) }

T_max:
(15) /* shortest time to reach upper bound */
(16) if (isInside){
(17)       // remaining distance to leave R2
(18)       d := - signedDist(R2, newPos);
(19) } else {
(20)       // remaining distance to enter R1
(21)       d := signedDist(R1, newPos);
(22) }
(23) return (d - err_pos) / v_max;

T_min:
(24) /* shortest time to reach lower bound */
(25) if (isInside){
(26)       // remaining distance to leave R1
(27)       d := - signedDist(R1, newPos);
(28) } else {
(29)       // remaining distance to enter R2
(30)       d := signedDist(R2, newPos)
(31) }
(32) return (d + err_next) / v_max;
```

Figure 4. Object-side processing of CRQ with distance-tolerance.

4.3 Update Policy

The algorithm in Figure 4 always defers a position fix such that the MO could cross the update region completely before acquiring the next position. However, the MO often does not actually move at maximum velocity. Then, it is located somewhere inside the update region at the time the next position fix is performed. This situation offers an important possibility to further optimize energy use. The larger the distance tolerance defined by the query, the more room for optimization can be utilized by the update policy.

An effective update policy must balance both factors driving the energy consumption. On the one hand, a high frequency of position fixes should be avoided. Accordingly, an update should be sent whenever this extends the forthcoming sleeping time of the sensor. On the other hand, sending an update message consumes a significant amount of energy as well. To reduce this factor, a message should only be sent if absolutely required for query correctness. For example, consider an object that changes its direction of movement at some time after entering the update region and moves back to where it came from. Then, no update message is required at all. In addition, deferring an update also causes fewer changes to the query result. This can avoid an oscillating result set when an MO moves back and forth around the query boundary and applications will be relieved from rapid result updates caused by such a movement.

Often, the right time to generate an update depends on the future movement – which is usually not known in advance. For that reason, we propose two alternative policies to balance between sensing and reporting operations. Their performance is then evaluated and compared against each other in Section 5.

0-7695-2947-X/07 $25.00

(1) Fraction-Delay FD(*f*): This policy decides to send an update message only when a certain fraction *f* of the update region is crossed *($f \in [0,1]$)*. This condition depends on the boundary from which the MO entered the update region. If the distance to this boundary exceeds the fraction *f* of the distance between both boundaries, an update message is generated. The reason is that no update is required if the MO leaves the update region at the same boundary, it came from. To evaluate this condition, we can reuse the computed time spans to reach both bounds (T_{min}, T_{max}). Recall that T_{min} becomes negative inside of the update region while T_{max} is still positive. Thus, an update is sent, iff:

$$-T_{min} > f \cdot (T_{max} - T_{min}) \quad (3)$$

With a larger update fraction, an update message is generated at a later point in time. In extreme ($f = 1$) an update is never generated in `updatePolicy`. Instead, it is only triggered by `mustUpdate` (Figure 4) when required to guarantee the tolerance constraints. Thus, less update messages are generated on average. However, a close distance to the query boundary in turn requires more position fixes.

(2) Predicted-Direction PD(*f, a*): The FD policy, although simple to implement, may not be suitable for all situations. As discussed, the best time to send an update depends on the future movement of the object. For that reason, the next policy extends FD by predicting the future direction of movement based on past position fixes in order to make a better decision. Specifically, it reports the location only if a certain fraction *f* of the update region is crossed and the movement is furthermore predicted to require an update message in the near future. This is determined by first computing the direction of reaching the respective boundary in the shortest time ($\vec{m}_{out}$). An object is considered to approach this boundary only if its predicted movement direction ($\vec{m}_{pred}$) deviates from $\vec{m}_{out}$ by an angle within $\pm a$ (where $0° \leq a \leq 180°$).

Given the MO's current location $loc(O_i)$, $\vec{m}_{out}$ can be determined as follows: let X_i denote the closest point to $loc(O_i)$, which is located on the upcoming query boundary – R_2 (R_1) if the object is currently (not) element of the result set. Then, $\vec{m}_{out}$ is the vector (X_i - $loc(O_i)$). Likewise, we compute the future direction of movement based on preceding (P_i) position fixes: $\vec{m}_{pred}$ then corresponds to the vector ($loc(O_i)$ - P_i). However, more sophisticated prediction algorithms [5],[14] could be used in practice as well.

To summarize, the introduction of tolerances offers three advantages for the processing of continuous range queries. First, it guarantees query results that are always correct within well-defined, application-specific bounds. As shown in Section 3, such guarantees could not be provided for non-tolerant range queries. Second, the energy spent on position sensing is reduced because the average waiting time between position fixes is increased and short sleeping times can be avoided. Finally, the amount of sensing and reporting operations can be balanced inside the update region, which can further reduce energy use. In order to identify the most effective balance, we evaluate the performance of the proposed update policies next.

5. Evaluation

In order to evaluate the performance of our approach, we have conducted various experiments based on a realistic mobility model. We will explain the experimental setup, followed by the detailed results.

5.1 Simulation Setup

We used the CanuMobiSim simulator [25] to generate movement traces of pedestrians following trip sequences through the inner city of Stuttgart. This comprises a simulation area of 2.0 x 2.0 km². Movements through the streets follow a smooth motion pattern [2]. This model uses stochastic principles to control the change of speed and direction in order to obtain a realistic movement behaviour. The target speed is chosen randomly from 0-3 m/s every 30 seconds. The sensing uncertainty is obtained from a statistical error model of GPS receivers [24] based on Gauss-Markow processes with an imprecision ≤ 6.3 m in 95 %. For each experiment we used 100 different movement traces along with five different measurement errors. Each result thus depicts the average of 500 runs. A single query is evaluated at a time with a lifetime of 3 hrs. For each object, the number of position fixes and updates is accumulated over the whole lifetime of a query. Then, we use these values to derive the overall energy consumption (of one object).

The algorithms assume a maximal sensing uncertainty of $S_{acc} = 10$ m, a maximal communication delay of $c_{max} = 1$ s and maximal velocity of $v_{max} = 5$ m/s, which reflects a pessimistic bound on actual speeds. Concerning energy costs, we assume that sending an update message consumes $W_U = 150$ mJoules. According to [9], this amount suffices to transmit about 240 bytes over GSM/GPRS. Each position fix is assumed to cost $W_S = 75$ mJoules and to take $T_{sense} = 0.5$ s. These are typical values of a low-power GPS receiver [18].

5.2 Energy consumption

To evaluate the energy consumption of d-CRQ for different sizes of the tolerance region, we varied both query boundaries (i.e., R_1 and R_2). Assume a rectangu-

lar region R with a size of 600 x 600 m² placed in the middle of the simulation area. Then, the inner region R_1 is obtained by shrinking R in each direction by d_r. Enlarging R in each direction by d_r likewise establishes the outer region R_2. The resulting width of the tolerance area is $2 \cdot d_r$. The minimal size to guarantee correct query results is $2 \cdot err_{next} = 35$m in this setup.

Figure 5 presents the evaluation of FD(f) policy for different values of d_r. It depicts the aggregated number of (a) position fixes, (b) updates and (c) the resulting energy consumption of each object in relation to the update fraction f. Most of all, we can observe a very different performance for update fractions smaller and larger than 0.5. If f is reduced below 0.5, the number of both, position fixes and updates increases. This can be explained as follows: First, recall that lower update fractions cause updates to be sent at smaller distances to the boundary the MO crossed already. Now, consider an object is moving into the outer region R_2 and sends an update message before its distance to R_2 exceeds the remaining distance to the upcoming boundary R_1 (i.e., $f < 0.5$). After the update is performed, the waiting time T_{wait} has to be computed based on R_2 instead of R_1. As the object is still located closer to R_2 than to R_1, this reduces the waiting time till the next position fix. In consequence, the total number of position fixes increases with smaller update fractions. Additionally, the object also has to re-evaluate the update decision after that position fix based on R_2. The smaller the preceding waiting time, the higher is the chance that the object is still located close to R_2 at that time. Thus, the update policy will trigger another update message immediately. The bottom line is that a low update fraction generates a series of frequent update messages (one after each position fix) while an object traverses the update region. This causes the number of updates to increase rapidly if f is reduced below 0.5. Notice that for such low values of f even more messages are sent for higher values of d_r because an MO must cross a larger distance before this series of rapid updates ends.

For $f > 0.5$, the number of position fixes also increases again (Figure 5a). This is due to shorter waiting times at locations close to the upcoming boundary. If an update is deferred beyond $f = 0.5$, this boundary becomes closer than the boundary that was crossed already. Yet, this can sometimes prevent an update message from being sent, if the object changes its direction of movement before reaching the respective boundary. For that reason, the number of update messages simultaneously drops slightly (Figure 5b).

As a result, we can see from Figure 5c that FD(0.5) permanently performs best for all depicted values of d_r. For smaller update fractions, too many updates are generated. For larger update fractions the increase in position fixes requires too much energy and dominates the savings in update messages.

Next, we added a movement prediction to FD(0.5) to further delay updates (as described in Section 4.3). Instead of using a large update fraction, an MO should not delay an update if it has a high chance of entering R_1 soon. The resulting performance of PD(0.5, a) with varying angles of tolerated deviation a is depicted in Figure 6. Smaller values of a successfully reduce the number of updates, as shown in Figure 6b. However, the simultaneous increase in position fixes (Figure 6a) is still too high and does not outweigh the reduction of update messages. Thus, the lowest energy is consumed

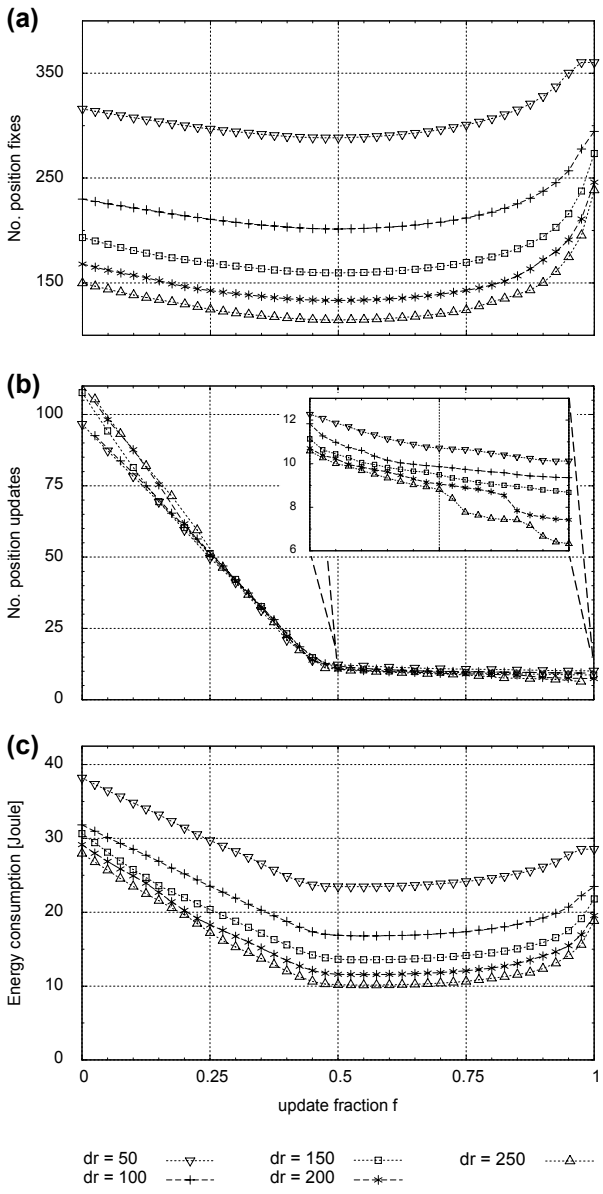

Figure 5. Evaluation of d-CRQ with FD(f) policy.

with the largest angle $a = 90°$ (Figure 6c), which in fact resembles the FD(0.5) policy. However, if sending an update is more costly – either in terms of energy or because a small number of update messages is compulsory – the direction prediction PD(0.5, *a*) with a small value of *a* offers an interesting alternative. It achieves to minimize the required communication while consuming less energy than a high update fraction.

After all, FD(0.5) is found to be the best policy for minimizing the amount of energy consumed for d-CRQ. Furthermore, Figure 5c and Figure 6c show that tolerances do help in reducing energy consumption. Increasing the update region (in terms of d_r) significantly reduces the amount of energy consumed. For comparison, we also evaluated the preliminary algorithm from Section 3, using the initial query region *R* (without tolerances) and obtained an energy consumption of about 50.5 Joules. In contrast, FD(0.5) with a minimal tolerance of $d_r = 17.5$ m guarantees correct query results and requires only 42.4 Joule. This constitutes a saving of 16%. With $d_r = 50$ m another 37% of energy is saved. In fact, we can even save up to 80% in total by further increasing the tolerance to $d_r = 250$ m. In short, this evaluation approves that increasing the distance tolerances can reduce the energy consumption significantly.

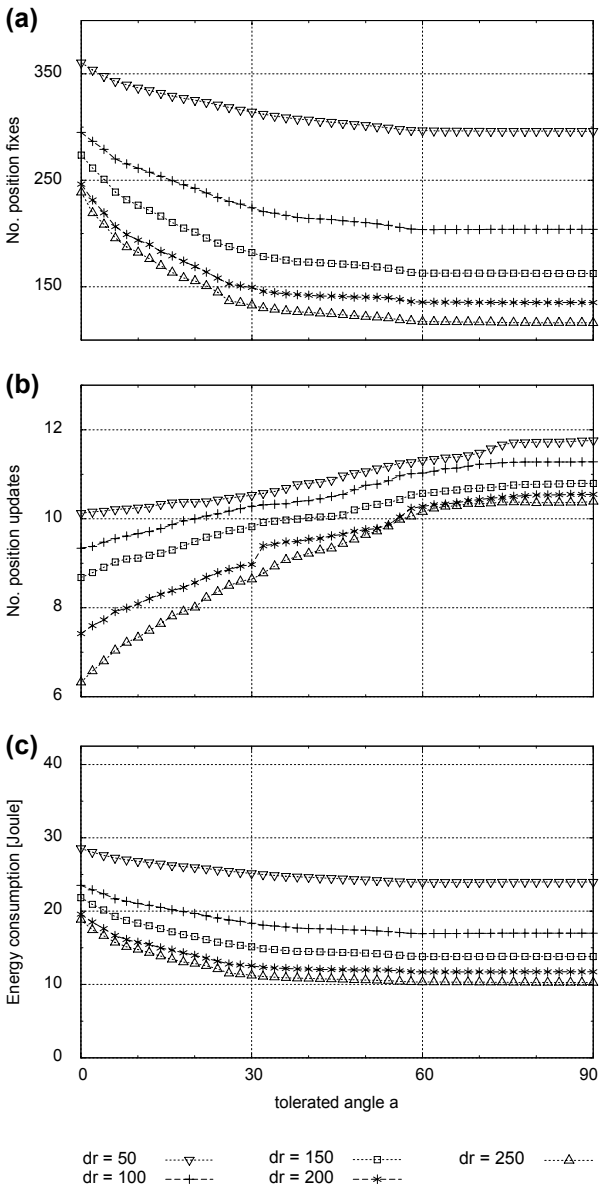

Figure 6. Evaluation of d-CRQ with PD(0.5, *a*) policy.

6. Related Work

The approach discussed in this paper can be classified as a *data stream filtering* technique. The objective of filtering is to facilitate the efficient evaluation of continuous queries over constantly-changing data streams (e.g., locations of mobile objects, temperature). Specifically, each stream source is installed with some constraints (called *filter conditions*) derived from query requirements. A data item generated at the source is sent to the central server only if its value satisfies the conditions defined in these filters. In our approach, the MO decides when to report an update, and so it acts as a "filter". It was shown in [1],[3],[22] that significant communication effort can be saved by assigning filter conditions appropriately.

To improve the performance of filters, the concept of *tolerance* has been proposed [20]. This assumes that users can tolerate some degree of imprecision (called *tolerance*) in query results. This tolerance is incorporated into filter conditions. A well-studied tolerance is the *value-based* tolerance, which is basically a numerical value for specifying the maximum error allowed. For example, filters are developed in [19] to answer tolerant average and minimum queries. In [1], filters are designed for top-*k* queries. In [11] a Kalman Filter is installed at every stream. The extension of filter methods in a sensor network is studied in [7]. Filters for non-value tolerances are developed in [4].

In these works, items generated by streams are assumed to be *always* correct. However, this assumption is not always valid. As we have explained in Section 3, the data acquired by a sensing device is contaminated with different kinds of errors. If these uncertainties are not considered, stream filters can miss important events, and introduce incorrectness into query results. In fact, the tolerance definitions described in the previous work are not "uncertainty-aware". Our work, on the other hand, specifies necessary conditions in tolerance definitions to ensure that they can be enforced with consideration of data uncertainty.

0-7695-2947-X/07 $25.00 © 2007 IEEE

Moreover, to the best of our knowledge, none of these stream filters considers the impact of *sensing*. They assume that communication is the only relevant factor of energy consumption and thus focus on minimizing the number of update messages. However, the energy costs of acquiring new sensor data can also have a significant impact on energy use – at least for the prominent sensing technology GPS. Our results in Section 5 confirm that there is an important trade-off between the amount of sensing and reporting operations required. Deferring an update as long as possible can increase the sensing costs significantly. In our paper, we carefully control this trade-off in order to reduce the overall energy consumption.

Recently, the issues of energy consumed for sensing have been considered in sensor networks. A good overview of approaches is given in [23]. Commonly, these solutions exploit correlations between values of multiple sensors – either on the same node [8],[15], or on multiple nodes in spatial proximity [6]. The consumed energy is reduced by acquiring data from a subset of sensors only and predicting the expected value of others with some level of confidence. This differs from the problem considered in this paper.

7. Conclusion

We studied how the energy of MOs can be saved when their locations are monitored by continuous range queries. We developed object-side processing algorithms that consider both energy use and the degree of correctness (or tolerance) in query results. These tolerance definitions are "uncertainty-aware", which consider various sources of data uncertainty. Our algorithms control the sensing and reporting operations carefully so that these tolerance definitions are satisfied. Moreover, our experiments show that the algorithms developed save energy significantly. In the future, we will extend this technique to other continuous queries (e.g., nearest-neighbour queries). We will also extend our algorithms to support concurrent execution of multiple queries and consider how other kinds of tolerances can improve energy savings.

Acknowledgments

The work described in this paper was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627, by the HKSAR Research Grants Council (RGC) CERG grant (PolyU 5138/06E), and by the Germany/Hong Kong Joint Research Scheme (DAAD PPP D/06/00383; G_HK013/06). We would also like to thank the anonymous reviewers for their insightful comments and suggestions.

References

[1] B. Babcock and C. Olston: "*Distributed Top-K Monitoring*". In: Proc. ACM Int'l Conf. Mgmt. of Data (SIGMOD'03), San Diego, USA, June 2003.

[2] C. Bettstetter: "*Smooth is Better than Sharp: A Random Mobility Model for Simulation of Wireless Networks*". In: Proc. 4th Int'l Work. Modeling, Analysis, and Simulation of Wireless and Mobile Syst. (MSWiM'01), Rome, Italy, July 2001.

[3] Y. Cai, K. Hua, G. Cao, and T. Xu: "*Real-Time Processing of Range-Monitoring Queries in Heterogeneous Mobile Databases*". In: IEEE Trans. Mobile Comp. 5(7), July 2006.

[4] R. Cheng, B. Kao, S. Prabhakar, A. Kwan, and Y. Tu: "*Adaptive Stream Filters for Entity-based Queries with Non-Value Tolerance*". In: Proc. 31st Int'l Conf. Very Large Data Bases (VLDB'05), Trondheim, Norway, Sep. 2005.

[5] A. Civilis, C. Jensen, and S. Pakalnis: "*Techniques for Efficient Road-Network-Based Tracking of Moving Objects*". In: IEEE Trans. Know. Data Eng. 17(5), May 2005.

[6] Y. Kotidis: "*Snapshot Queries: Towards Data-Centric Sensor Networks*". In: Proc. 21st Int'l Conf. Data Engineering (ICDE'05), Apr. 2005.

[7] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos: "*Processing Approximate Aggregate Queries in Wireless Sensor Networks*". In: Inf. Syst. Jour. 31(8), Dec. 2006.

[8] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong: "*Model-driven data acquisition in sensor networks*". In: Proc. 30th Int'l Conf. Very Large Data Bases (VLDB'04), Toronto, Canada, Sep. 2004.

[9] B. Gedik and L. Liu: "*MobiEyes: A Distributed Location Monitoring Service Using Moving Location Queries*". In: IEEE Trans. Mobile Comp. 5(10), Oct. 2006.

[10] M. Gruteser and D. Grunwald: "*Anonymous Usage of Location-Based Services through Spatial and Temporal Cloaking*". In: Proc. 1st Int'l Conf. Mobile Systems, Applications, and Services (MobiSys'03), San Francisco, USA, May 2003.

[11] A. Jain, E. Chang, and Y. Wang: "*Adaptive stream resource management using Kalman Filters*". In: Proc. ACM Int'l Conf. Mgmt. Data (SIGMOD'04), Paris, France, June 2004.

[12] C. Jensen, A. Friis-Christensen, T. Pedersen, D. Pfoser, S. Saltenis, and N. Tryfona: "*Location-based services – A database perspective*". In: Proc. 8th Scand. Conf. Geo. Inf. Science (ScanGIS'01), Ås, Norway, 2001.

[13] A. Leonhardi and K. Rothermel: "*Architecture of a Large-scale Location Service*". In: Proc. 22nd Int'l Conf. Distr. Comp. Syst. (ICDCS'02), Vienna, Austria, July 2002.

[14] A. Leonhardi, C. Nicu, and K. Rothermel: "*A Map-based Dead-reckoning Protocol for Updating Location Information*". In: Proc. Int'l Parallel and Distr. Processing Symp. (IPDPS'02), Ft. Lauderdale, USA, 2002.

[15] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. "*The design of an acquisitional query processor for sensor networks*". In: Proc. ACM Int'l Conf. Mgmt. Data (SIGMOD'03), San Diego, USA, June 2003.

[16] P. Misra and P. Enge: "*Global Positioning System: Signals, Measurements and Performance*", 2nd Ed., Ganga-Jumuna Press, 2006.

[17] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao: "*A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors*". In: IEEE Trans. Know. Data Eng. 17(11), Nov. 2005.

[18] Navman: *Jupiter 30 Data sheet*, http://www.navman.com/Documents/OEM_docs/Jupiter 30/LA000576-C_Jupiter30_DataSheet.pdf, May 2007.

[19] C. Olston, J. Jiang, and J. Widom: "*Adaptive Filters for Continuous Queries over Distributed Data Streams*". In: Proc. ACM Int'l Conf. Mgmt. Data (SIGMOD'03), San Diego, USA, June 2003.

[20] C. Olston and J. Widom: "*Efficient Monitoring and Querying of Distributed, Dynamic Data via Approximate Replication*". In: IEEE Data Eng. Bull. 28(1), Mar. 2005.

[21] D. Pfoser and C. Jensen: "*Capturing the Uncertainty of Moving-Object Representations*". In: Proc. 6th Int'l Symp. Spatial Databases (SSD'99), Hong Kong, 1999.

[22] S.Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch: "*Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects*". In: IEEE Trans. Comp. 51(10), Oct. 2002.

[23] V. Raghunathan, S. Ganeriwal, and M. Srivastava: "*Emerging techniques for long lived wireless sensor networks*". In: IEEE Comm. Mag. 44(4), Apr. 2006.

[24] J. Rankin: "*GPS and Differential GPS: An Error Model for Sensor Simulation*". In: IEEE Position Location and Navigation Symp. (PLANS'94), Las Vegas, USA, Apr. 1994.

[25] I. Stepanov, P. Marron, and K. Rothermel: "*Mobility Modeling of Outdoor Scenarios for MANETs*". In: Proc. 38th An. Simulation Symp. (ANSS'05), San Diego, USA, Apr. 2005.

[26] U. Varshney: "*Location management for mobile commerce applications in wireless internet environment*". In: ACM Trans. Internet Tech. 3(3), Aug. 2003.

[27] O. Wolfson, A. Sistla, S. Chamberlain, and Y. Yesha: "*Updating and Querying Databases that Track Mobile Units*". In: Distributed and Parallel Databases, 7(3), July 1999.

0-7695-2947-X/07 $25.00 © 2007 IEEE