Processing Continuous Range Queries with Spatiotemporal Tolerance

Tobias Farrell, Kurt Rothermel, and Reynold Cheng, *Member*, *IEEE*

Abstract—Continuous queries are often employed to monitor the locations of mobile objects (MOs), which are determined by sensing devices like GPS receivers. In this paper, we tackle two challenges in processing continuous range queries (CRQs): coping with data uncertainty inherently associated with location data, and reducing the energy consumption of battery-powered MOs. We propose the concept of *spatiotemporal tolerance* for CRQ to relax a query's accuracy requirements in terms of a maximal acceptable error. Unlike previous works, our definition considers tolerance in both the spatial and temporal dimensions, which offers applications more flexibility in specifying their individual accuracy requirements. As we will show, these tolerance bounds can provide well-defined query semantics in spite of different sources of data uncertainty. In addition, we present efficient algorithms that carefully control when an MO should sense or report a location, while satisfying these tolerances. Thereby, we particularly reduce the number of *position sensing* operations substantially, which constitute a considerable source of energy consumption. Extensive simulations confirm that the proposed algorithms result in large energy savings compared to nontolerant query processing.

Index Terms—Tracking mobile objects, continuous queries, distributed processing, data uncertainty, energy consumption.

1 Introduction

Driven by the ongoing advances in wireless communication and sensing technologies, *context-aware systems* have emerged as an important class of mobile computing. In those systems, the location of entities has been identified as primary context [6], and many applications rely on real-time location information of a potentially large number of mobile objects (MOs).

In such systems, each MO is equipped with a positioning sensor (e.g., a GPS receiver) and reports its position over a wireless connection (e.g., a GSM/GPRS network) to a so-called location manager (LM). At the LM, location-aware applications can register *continuous queries* [22] to monitor the activities of MOs. In contrast to one-time queries, such queries reside in the system and are continuously evaluated for an extensive period. In this paper, we focus on one important class of such queries: the continuous range query (CRQ). It returns all MOs located inside some spatial region. That is, the query result is updated whenever an MO enters or leaves this region. CRQs are widely used in context-aware applications to identify all MOs in a certain spatial context. Examples include sending personalized advertisements or e-coupons to customers entering the vicinity of a store [28], monitoring traffic conditions of road sections, and generating alerts when your children, pets, or bicycles leave a safety zone [31].

- T. Farrell and K. Rothermel are with the Institute of Parallel and Distributed Systems, Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany. E-mail: {tobias.farrell, kurt.rothermel}@ipvs.uni-stuttgart.de.
- R. Cheng is with the Department of Computer Science, University of Hong Kong, CB327, Chow Yei Ching Building, Pokfulam Road, Hong Kong. E-mail: ckcheng@cs.hku.hk.

Manuscript received 8 Oct. 2008; revised 11 Sept. 2009; accepted 23 June 2010; published online 26 Aug. 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2008-10-0398.
Digital Object Identifier no. 10.1109/TMC.2010.159.

Efficient processing of continuous queries has attracted a lot of research interest lately. In particular, the idea of *object-side processing* has been utilized in various projects [3], [9], [18], [29]. They leverage the processing capabilities of mobile devices to evaluate parts of a query locally on each MO. That is, when a new CRQ is registered, the LM first propagates the query region to all MOs. Then, each MO monitors its movement locally and sends a report to the LM only when it crosses the region's boundary. In doing so, no reports are required while an MO moves without affecting the query result. This can save a large number of communication messages, which in turn reduces the server-side workload as well as the energy consumption of MOs [3], [9].

The latter is of particular importance, as many small, mobile devices (like cell phones and PDAs) are battery powered, and energy is their most precious resource. Maximizing the lifetime of those devices is thus essential for user acceptance. While most previous works focus solely on communication costs [3], [9], [18], [29], we argue that the energy consumption of *position sensing* often cannot be ignored either [8], [23]. Even in low-power mode, a common GPS receiver consumes no less than 75 mJoules to acquire a new position [19]. The same amount of energy is required by GPRS to transmit about 120 bytes of data [9]. Moreover, object-side processing typically requires much more sensing than reporting operations. In particular, we will show that there is an important trade-off between the frequency of sensing and the accuracy of query results.

In this paper, we therefore study how sensing costs can be reduced by *relaxing* a query's accuracy requirements. This is motivated by the observation that many applications do not require the highest degree of accuracy, but can often tolerate some well-bounded error—in exchange for lower energy costs. For example, assume a CRQ is used for distributing warning messages within a spatial region. It may be well acceptable (or even desirable) that some users receive the alert early (before entering the query region). In contrast, for distributing location-based advertisements inside a supermarket, some users could receive it late (after entering the store). To capture diverse accuracy requirements, we introduce the concept of *spatiotemporal tolerance* (or st-tolerance). It gives applications the flexibility to define their notion of acceptable errors precisely. They can define tolerances not only in the *spatial* domain, but also in terms of *time*. For example, the spatial tolerance can be that MOs close to the region (e.g., within a distance *d*) may be returned as well. With time tolerance, one can express that MOs have to be added to the result set only after residing inside the query region for a certain period.

Moreover, st-tolerance helps to cope with *data uncertainty* inherently associated with location data [21]. Due to technical limitations, like sensing inaccuracy and communication delay, the result of a nontolerant CRQ may always deviate from the actual result observed in reality, and there are no conceptual means for informing the query user about the degree of deviation. In contrast, we show how the semantics of *tolerant CRQ* can be provided despite data uncertainty. That is, we limit the deviation from the actual result to those MOs inside the spatial and/or temporal range defined by the tolerance. This allows applications to control precisely, which kind of deviation they are willing to accept. For example, it could tolerate some MOs in the region's proximity (false positives), but not accept any MOs inside the store to be missed (false negatives).

To achieve this, we develop efficient algorithms to schedule sensing and reporting operations of an MO, and we derive *minimal requirements* for the tolerance to be *satisfiable* in consideration of different sources of uncertainty. With a minimal tolerance, our algorithms ensure query results that are *correct within the given tolerance*. Accepting larger tolerances, furthermore, allows trading query accuracy off against energy use. In contrast to previous works, we thereby focus particularly on reducing *sensing* costs. Yet, this is achieved without increasing *communication* costs. In fact, we will show that the number of reports is further reduced as well when handling multiple, simultaneous CRQs.

We have previously proposed an initial solution using distance-based tolerance [7]. Here, we extend this approach by defining tolerances along both the spatial and the temporal dimensions. Moreover, we now discuss processing multiple queries efficiently, and present considerably extended evaluation results, assessing not only the resulting energy consumption, but also the provided accuracy of query results. While this paper focuses on continuous range queries over mobile objects, we are convinced that the main concepts are relevant to other types of continuous queries and other domains of sensor data as well (e.g., [18], [20]). In summary, our contributions are:

- proposing the definition of spatial and spatiotemporal tolerance semantics for CRQ,
- deriving minimum tolerance constraints to satisfy the tolerance semantics in spite of data uncertainty,
- developing efficient algorithms that can cope with data uncertainty and reduce the total energy consumed by sensing and reporting operations, and
- verifying the effectiveness of our approach by extensive simulation using a comprehensive mobility model.

The remainder of this paper is organized as follows: In Section 2, we first describe our system model. The formal definition of st-tolerance is presented in Section 3. Subsequently, Section 4 discusses the benefits of processing CRQs with tolerance step by step. Then, we evaluate the performance of our approach in Section 5, discuss related work in Section 6, and conclude the paper in Section 7.

2 System Model

Our system model consists of mobile objects (MOs) and a location manager (LM). MOs represent any mobile, energy-constrained devices (like cell phones or PDAs) equipped with a processor, a wireless network interface, and a positioning sensor. Each MO is identified by a globally unique id (like an IP address or phone number), denoted by o_i.

The LM processes CRQs on behalf of location-aware applications. A CRQ is defined by a fixed query boundary *qb* that confines a closed spatial region. Applications can register a CRQ at the LM to monitor a region of interest. The query then remains *active* until it is deregistered again. During that time, the CRQ should always return the ids of all MOs currently located inside *qb*. That is, after returning an initial result, the LM must provide (differential) updates, whenever an MO crosses *qb*.

We do not assume any particular realization of the LM. It may be distributed over multiple servers, to which the MOs are mapped (dynamically) [14], [29]. Each MO communicates with a single LM-server over a wireless network (like GPRS, UMTS, or Wi-Fi meshes). To participate in query processing, MOs have to perform two different operations that require further consideration.

2.1 Position Sensing

First, an MO has to acquire its geographic position locally. Typically, this comprises some complex sensor activity, which consumes a certain amount of energy. For example, GPS requires about 0.5 s to compute a position fix based on pseudorange measurements of satellite signals [16]. Likewise, WLAN-based positioning requires a series of signal strength measurements [10]. While no position is required, the sensor can, however, remain in a low-power *sleep mode* to conserve energy[1] [8], [23]. In the following, we assume that each sensing operation is invoked explicitly by the MO and requires a known time τ_{sense} to complete. An MO's true position at time t is denoted by $pos(o_i, t)$. But due to limited sensing accuracy, the returned sensor position can deviate from $pos(o_i, t)$ to a certain extent. In this regard, we assume a known upper bound δ_{sense} of the deviation, which is a characteristic sensor property.

2.2 Communication

In addition, an MO has to send report messages to the LM from time to time. As all reports will be similar in size, we

1. In this mode, a GPS receiver may still wake up at low frequency to refresh satellite data and prevent long start-up times (cf. "Push-To-Fix" mode in [19]). However, such background energy is consumed independently of the frequency of sensing operations.

TABLE 1
Symbols Used in the Paper

Symbol	Description
o_i	Unique id of the i-th MO ($1 \leq i \leq n$)
$pos(o_i, t)$	True position of the i-th MO at time t
τ_{sense}	Time to perform a sensing operation
δ_{sense}	Max. deviation between sensed and true position
τ_{com}	Max. communication delay between MO and LM
$v_{\max}$	Max. velocity of an MO

assume that each message requires the same amount of energy. Moreover, the communication network causes some delay before a report reaches the LM. We assume a known upper bound τ_{com} of the end-to-end communication delay.[2] While mechanisms for ensuring a maximum delay have been proposed (cf. [5]), existing network technologies often have unbounded delays. However, it was shown that reasonable upper delay bounds that hold with high probability can be empirically determined based on the networking environment [17].

Besides communication and position sensing, an MO also has to perform some local processing. Since this involves only simple computations, the processing delay and energy consumption are negligible. Finally, we assume that each MO has knowledge about its maximum 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]. We do not assume movements to be limited to a road network. Thus, we use the euclidean distance to measure how far two positions are apart. Table 1 summarizes the symbols used throughout this paper.

3 Tolerant CRQ: Definition and Semantics

In this section, we propose our concept of a tolerant CRQ. By specifying a so-called *tolerance space*, applications can control how the result of a tolerant CRQ may deviate from the actual result. This tolerance space "blurs" the query boundary qb in both the space and time dimension, which relaxes the accuracy requirements for those MOs inside the tolerance space. As mentioned above, this provides for query results that are *correct within the given tolerance*.

For clarity of presentation, let us first consider the spatial dimension of a tolerance space. Then, we extend this scheme to support the concept of temporal tolerance.

3.1 Spatial Tolerance

Spatial tolerance of a CRQ is based on spatial proximity to qb. In particular, we introduce two boundaries that sandwich qb. The *inner* query boundary qb_{I} is located inside qb (denoted by $qb_{\text{I}} \subseteq qb$), while the *outer* one qb_{O} is located outside qb (i.e., $qb \subseteq qb_{\text{O}}$). Both can be of arbitrary shape. The area enclosed by these two boundaries defines the tolerance space. For MOs located inside this area (i.e., close to qb), it is not guaranteed whether they are included in the result.

2. In this paper, we focus on the correctness of query results maintained at the LM. If the transfer of query results to an application causes an extra delay, only τ_{com} must be adapted accordingly. This does not affect the algorithms presented in this paper.

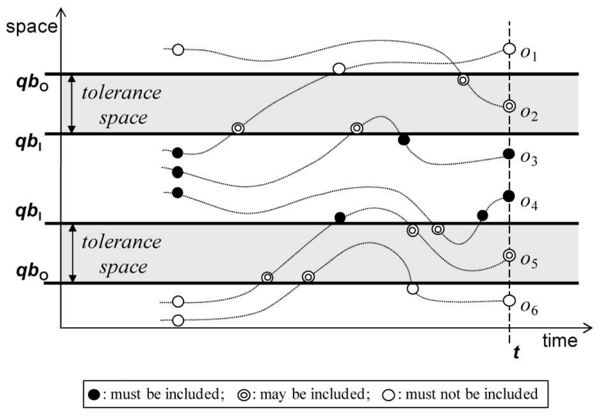


Fig. 1. CRQ with spatial tolerance.

Definition 1 (Spatially tolerant CRQ (s-CRQ)). *Let qb_{I}, qb_{O} be two closed boundaries, with $qb_{\text{I}} \subseteq qb_{\text{O}}$. Then, the result of an s-CRQ includes all MOs* currently *located inside qb_{I} but no MO* currently *located outside qb_{O}. That is, at any time t, the result is defined by the set $S \cup T$, with*

$$S := \{o_i | \; pos(o_i, t) \subseteq qb_{\text{I}}\}, \text{ and}$$
$$T \subseteq \{o_i | pos(o_i, t) \subseteq qb_{\text{O}}\} \; (1 \leq i \leq n).$$

According to this definition, the query result must contain all MOs that are located inside qb_{I} (the set S). However, MOs located inside the tolerance space might be contained in the result set, too. These objects belong to the set T, which is any subset of MOs located inside qb_{O}—including those outside qb_{I}. However, no MO located outside qb_{O} is contained in the query result. This is illustrated in Fig. 1, which shows the trajectories of several MOs in a one-dimensional space. At time t, the MOs o_1 and o_6 are located outside qb_{O}, and hence are not included in the query result. On the other hand, o_3 and o_4 must be included as they are located inside qb_{I} at that time. The MOs located in the tolerance space, o_2 and o_5, may be included in the result set.

Note that Definition 1 is independent of the original boundary qb. An application can choose qb_{I} and qb_{O} individually and thereby control the amount of false positives and false negatives in the query result. Fig. 2 shows three different settings of qb_{I} and qb_{O}. In the setting in Fig. 2a, the application has to cope with both false positives and false negatives, while Fig. 2b and Fig. 2c present special cases, avoiding either false negatives or false positives.

For example, assume that a ski resort's information system provides a tracking service to improve the safety of

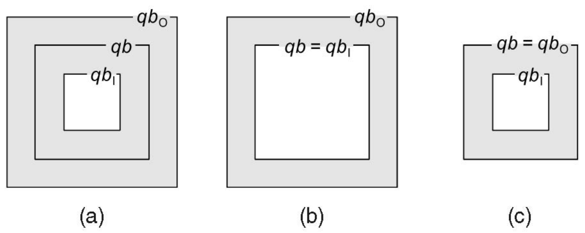


Fig. 2. Different configurations of s-CRQ. (a) False positives and false negatives. (b) False positives only. (c) False negatives only.

ski tourists, using their mobile phones to locate and warn them. CRQs are employed to send warning messages to skiers in regions that are considered critical in terms of avalanches. While these CRQs are required to return the ids of all MOs in the critical regions, we can certainly tolerate that some MOs in the regions' proximity are also included in the result set. These semantics can be achieved by setting $qb_{\rm I}$ to the critical region and $qb_{\rm O}$ to a size that meets the application's notion of proximity.

Next, assume we are also monitoring the region of the ski kindergarten to notify the instructor whenever a child is about to leave. In that case, false positives are not acceptable and thus $qb_{\rm O}$ needs to model the kindergarten's geographic boundary. However, we can set $qb_{\rm I}$ to a smaller area, tolerating that notifications are already generated while a child is approaching the outer boundary.

3.2 Spatiotemporal Tolerance

To introduce the temporal dimension, we again define the tolerance space indirectly by giving criteria for those MOs that must be included or excluded from the query result. The major difference to s-CRQ is that these criteria consist of both spatial and temporal constraints. The temporal constraint is based on the concept of a (temporal) *proximity window*. This includes all elapsed points in time that are considered to be "in proximity" to the current time. That is, a proximity window of length t, denoted as $<t>$, is defined by the time interval $[t_c - t, t_c]$, where t_c is the current time.

Now, we can extend the criteria for MOs to be included in the result set by a temporal constraint: "all MOs that have been *continuously* located in $qb_{\rm I}$ *during* $<t_{\rm I}>$." Consequently, an MO residing in $qb_{\rm I}$ for less than t_I time units is still located in the tolerance space, and hence may be subject to a false negative. Analogously, the criterion for excluding MOs is extended to "all MOs that have been *continuously* located outside of $qb_{\rm O}$ *during* $<t_{\rm O}>$." This leads to the following definition of a spatiotemporal CRQ.

Definition 2 (Spatiotemporally tolerant CRQ (st-CRQ)). *Let $t_{\rm I}, t_{\rm O} \geq 0$ be the length of two proximity windows and $qb_{\rm I}$, $qb_{\rm O}$ be two closed boundaries, with $qb_{\rm I} \subseteq qb_{\rm O}$. Then, the result of an st-CRQ includes all MOs that have been located in $qb_{\rm I}$ during the entire window $<t_{\rm I}>$ but no MO that has been located outside $qb_{\rm O}$ during the entire window $<t_{\rm O}>$. Thus, an st-CRQ always returns the set $S \cup T$, with*

$$S := \{o_i \mid \forall\, t \in <t_{\rm I}>: pos(o_i, t) \subseteq qb_{\rm I}\}, \text{ and}$$
$$T \subseteq \{o_i \mid \exists\, t \in <t_{\rm O}>: pos(o_i, t) \subseteq qb_{\rm O}\}\ (1 \leq i \leq n).$$

According to this definition, the query result must always contain set S, which includes all MOs continuously located inside $qb_{\rm I}$ in proximity window $<t_{\rm I}>$. The result set may also contain MOs inside the tolerance space spanned by the boundaries $qb_{\rm I}$, $qb_{\rm O}$ and proximity windows $<t_{\rm I}>$, $<t_{\rm O}>$. These MOs belong to set T, which is any subset of MOs that have been located in $qb_{\rm O}$ at least once during $<t_{\rm O}>$. But no MO that has been continuously located outside $qb_{\rm O}$ during window $<t_{\rm O}>$ is contained in the result. Thus, only those MOs that were spatially close to qb (i.e., entered $qb_{\rm O}$) in the recent past (i.e., during $<t_{\rm O}>$) may be included.

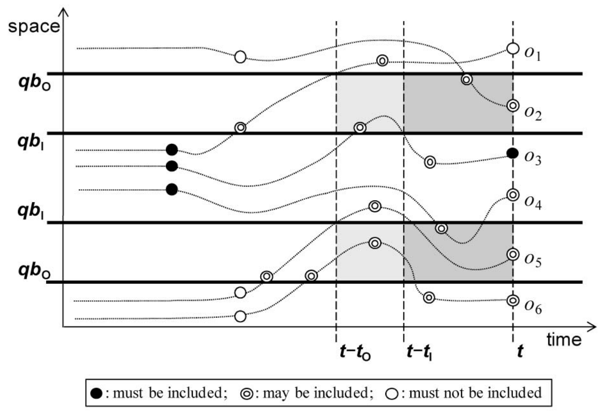


Fig. 3. CRQ with spatiotemporal tolerance.

This is illustrated in Fig. 3, which shows the same trajectories as Fig. 1. To determine the result set of the corresponding st-CRQ at time t, we have to consider three cases: 1) MOs that are inside $qb_{\rm I}$ at time t may be included in the result, if they were located outside $qb_{\rm I}$ at least once during $<t_{\rm I}>$, otherwise they are definitely included. Therefore, the MO o_4 may be included, while o_3 is definitely included in the result. 2) MOs that are outside of $qb_{\rm O}$ at time t may be included in the result, if they were located inside $qb_{\rm O}$ at least once during $<t_{\rm O}>$, otherwise they must be excluded. Therefore, o_6 may be in the result, whereas o_1 is not. 3) Finally, MOs that are inside $qb_{\rm O}$ but outside $qb_{\rm I}$ at time t may be included in the result set or not. Therefore, o_2 and o_5 may also be included as well. Apparently, st-CRQ further relaxes the inclusion and exclusion criteria. The spatiotemporal tolerance space depicted in Fig. 3 includes two more MOs (o_4 and o_6) than the tolerance space in Fig. 1.

Altogether, st-tolerance offers more flexibility and a finer level of control to specify which MOs may be subject to false positives or false negatives. Let us again consider the ski resort example above, where we add a new service for estimating the current waiting time at each of the resort's ski lifts. CRQs are used to determine the number of skiers currently waiting in a lift's waiting area. Since skiers may just pass this area, they are considered to wait only if they stay there longer than a certain threshold. This knowledge can be exploited to relax the corresponding st-CRQ by setting the temporal tolerance $t_{\rm I}$ to that threshold, while $qb_{\rm I}$ corresponds to the boundary of the waiting area and $qb_{\rm O}$ is set according to the minimal spatial tolerance possible. Therefore, the result set will include all skiers waiting in the corresponding area and potentially some more just passing this area. Adding the time constraint not only leads to estimates that are more precise, but also allows for reducing the energy consumption of MOs significantly, as will be seen below.

4 Processing Tolerant CRQs

In the following, we first present a basic algorithm to handle nontolerant CRQs. This allows us to identify the inherent problems of processing CRQs without tolerance. Then, we show how to overcome these shortcomings by extending

```
on obtaining position(pos) do:
         /* check reporting condition */
(1)      if (pos  inside qb AND lastReport ≠ "enter") then {
(2)           sendReport(id, pos, "enter");
(3)           lastReport := "enter";
         } else
(4)      if (pos outside qb AND lastReport = "enter") then {
(5)           sendReport(id, pos, "leave");
(6)           lastReport := "leave";
         }
(7)      start new sensing operation    // takes time $\tau_{sense}$ to complete
```

Fig. 4. Basic algorithm for object-side processing of a CRQ.

the algorithm to support st-tolerance. Again, we first consider a spatially tolerant CRQ and then extend our solution to spatiotemporal tolerance. In addition, we limit the discussions to a single active query until presenting optimizations for multiple concurrent queries in Section 4.5.

4.1 Object-Side Processing—Basic Algorithm

Let us first discuss the efficient processing of a nontolerant CRQ. For that purpose, we employ the concept of *object-side processing* [3], [9], [18], [29] as it requires the least reporting costs. Whenever a CRQ is registered, the LM sends an initialization message containing the query boundary *qb* to all MOs. Those MOs located inside *qb* immediately respond with an *enter* report, signaling that they belong to the initial result set. Subsequently, each MO monitors its own position locally and sends a new report (either *enter* or *leave*) whenever it crosses *qb*. This approach effectively minimizes the reporting costs, as reports are sent only if an MO actually affects the query result. The energy required for receiving initialization messages is easily amortized by saving report messages throughout the query's lifetime [3], [9].

Fig. 4 shows the basic algorithm performed by each MO. Whenever the MO obtains a new position from the sensor, it checks—based on the sensed value—whether it crossed *qb* since its previous report and, if so, generates a new one. Then, it immediately starts sensing again. As a result, the MO performs sensing with maximum frequency ($1/\tau_{\text{sense}}$), which is prohibitively costly in terms of energy. Nevertheless, this fails to guarantee correct query results at the LM at all times. Due to limitations of sensor technologies and communication latency, the query result may always deviate from the *actual result* observed in reality. Specifically, three different sources of data uncertainty add to this problem. Together they determine the worst-case error, which also plays an important role below for processing tolerant CRQs correctly.

1. **Sensing Uncertainty.** As discussed in Section 2.1, the sensed position can deviate from the actual position up to a distance of δ_{sense} due to limited sensor accuracy (as depicted by the shaded circles in Fig. 5).
2. **Sampling Uncertainty.** In addition, the resolution of sensing is limited by the time required for each sensing operation (τ_{sense}). For example, assume a sensed position is located just in front of *qb* and thus does not trigger a report. The MO may then move as far as $\tau_{\text{sense}} \cdot v_{\max}$ before the crossing event can be detected (as depicted by p_2 in Fig. 5).
3. **Communication Delay.** Finally, the report message arrives at the LM only after a certain delay ($\leq \tau_{\text{com}}$).

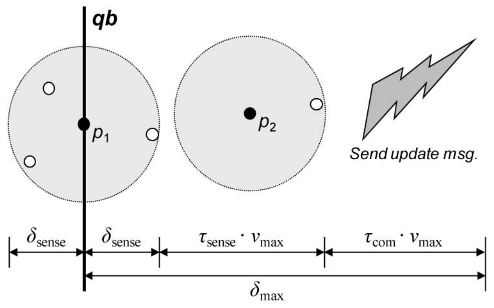


Fig. 5. Maximum error in updating the query result of CRQ.

While the report is transmitted over the network, the MO may move a further distance of up to $\tau_{\text{com}} \cdot v_{\max}$. In total, the MO's true position may hence be located at a distance $\delta_{\max}$ beyond *qb*, when the query result is updated at the LM (cf. Fig. 5), with:

$$\delta_{\max} = \delta_{\text{sense}} + (\tau_{\text{sense}} + \tau_{\text{com}}) \cdot v_{\max}. \tag{1}$$

While covering this distance, the MO is either a false positive or a false negative. It is important to notice that $\delta_{\max}$ depends only on the underlying technologies and is not a shortcoming of the presented algorithm. Sending a report any later would clearly increase the worst-case error even further. If a report was generated any earlier (i.e., based on some movement prediction), however, an MO could change its direction of movement (unpredictably) before actually crossing *qb*. This would require a second report soon after, which results in a larger worst-case error, too.

Next, let us consider the energy consumption of an MO. While the algorithm in Fig. 4 generates only a minimal number of reports, it performs sensing at highest frequency. To reduce the sensing costs, we can apply a simple optimization called *selective sensing* [8]: After acquiring a position, the MO can suspend sensing as long as it does not possibly affect the query result. This is the minimal time required to reach *qb* based on its maximal velocity ($v_{\max}$). Accordingly, we can replace line 7 of the previous algorithm (Fig. 4) as shown in Fig. 6. This decreases the number of sensing operations substantially without affecting the result accuracy. Yet, when the MO is close to *qb* the sensing rate remains very high. Whenever an MO approaches *qb* with less than $v_{\max}$, it must perform sensing in successively decreasing intervals, as the sensor suspension time shrinks along with the remaining distance to *qb* (line 7 in Fig. 6). Consequently, an MO still spends a substantial amount of energy on frequent position sensing in order to detect the crossing of *qb* on time. Reducing the sensing rate would decrease the accuracy of query results even further.

```
         /* low-power mode:  */
(7)      $t_{min}$  := (distance(pos, qb) / $v_{max}$);
(8)      $t_{susp}$ := max(0, $t_{min}$ - $\tau_{sense}$);        // $t_{min}$ < $\tau_{sense}$ close to qb

(9)      schedule next sensing operation after $t_{susp}$;
```

Fig. 6. Extending the algorithm in Fig. 4 with selective sensing.

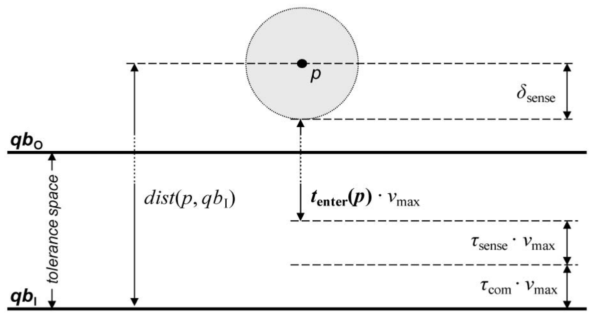

p
δ_{sense}
qb_O
tolerance space
$dist(p, qb_I)$
$t_{\text{enter}}(p) \cdot v_{\max}$
$\tau_{\text{sense}} \cdot v_{\max}$
$\tau_{\text{com}} \cdot v_{\max}$
qb_I

Fig. 7. The time until the MO must start sensing again.

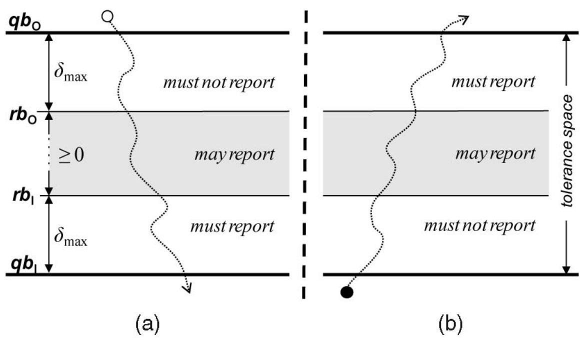


Fig. 8. Reporting decision inside the tolerance space. (a) "Enter" case. (b) "Leave" case.

In summary, nontolerant CRQs thus face two inherent problems: First, data uncertainty prevents precise query results and applications cannot control the resulting imprecision. At the same time, MOs try to provide the best possible accuracy, as they are unaware of the application's actual requirements. Even with selective sensing, this costs a lot of energy due to frequent sensing. We now discuss how this can be improved by utilizing the concept of tolerance.

4.2 Spatial Tolerance

In the following, we still employ the concept of object-side processing. Yet, the LM now propagates both query boundaries qb_I, qb_O (and for st-CRQ also t_I, t_O) to the MOs when a new query is registered. Each MO then utilizes the given tolerance space to provide for query results that are correct within the given tolerance and to reduce sensing costs.

According to the definition of s-CRQ, all MOs inside the inner region qb_I must be included in the result set, while all MOs outside the outer boundary qb_O must be excluded. Essentially, we can guarantee these semantics by updating the query result *early*, i.e., while an MO crosses the tolerance space. That is, an MO is added to the result set after entering qb_O but before entering qb_I and is removed after leaving qb_I but before leaving qb_O.

Let us consider how to schedule sensing operations to achieve this despite data uncertainty. Assume that an MO is located outside qb_O with a sensed position p (as shown in Fig. 7) and is currently not part of the query result. This MO can defer sensing, but must ensure that a potential enter report will *arrive* at the LM before itself crosses qb_I. Therefore, it has to start sensing again before coming too close to qb_I. In particular, let $\text{dist}(p, qb_I)$ denote the shortest distance to qb_I, then the MO must start sensing again after a duration

$$t_{\text{enter}}(p) = (\text{dist}(p, qb_I) - \delta_{\max})/v_{\max}. \quad (2)$$

Fig. 7 illustrates the various data uncertainties, summing up to $\delta_{\max}$ (cf. (1)), which have to be taken into account. A sensing operation that is initiated after t_{enter} and takes τ_{sense} definitely completes while the MO's distance to qb_I is larger than $\tau_{\text{com}} \cdot v_{\max}$. Thus, a report generated at that time still arrives at the LM in time. Starting sensing any later could, however, result in violating the query semantics.

After obtaining a new position p' from the sensor, the MO can then decide to send a report right away or to schedule another sensing operation first. Any $t_{\text{enter}}(p') \geq 0$ clearly allows for another sensing operation, but the MO could still send a report earlier. To explain this reporting decision, let us define the following *reporting boundaries* (see Fig. 8a):

- The **inner reporting boundary rb_I** is located at a distance $\delta_{\max}$ outside qb_I (i.e., it is obtained by enlarging qb_I in each direction by $\delta_{\max}$).
- The **outer reporting boundary rb_O** is located at a distance $\delta_{\max}$ inside qb_O (i.e., it is obtained by shrinking qb_O in each direction by $\delta_{\max}$).

The MO uses these boundaries to distinguish three cases when making the reporting decision (see Fig. 8a):

1. If p' is *inside* rb_I, the MO **must report** "enter" immediately. In this case, $t_{\text{enter}}(p') < 0$ holds. Thus, starting another sensing operation without reporting could cause a violation of the query semantics.
2. If p' is *outside* rb_O, the MO **must not report** "enter." That is, it may only send an enter report if p' is at a distance larger than $\delta_{\max}$ inside qb_O. This constraint is required to ensure that a subsequent "leave" event can be detected and transmitted to the LM in time (before leaving qb_O again).
3. If p' is *between* rb_I and rb_O, the MO **may report** "enter" (and may also schedule another sensing operation first). We call this the *may-report zone*. The reporting decision in this zone may influence the resulting performance significantly, and we discuss a suitable *reporting policy* in Section 4.4.

Analogically, an MO included in the result set must ensure to generate a leave report using interchanged roles of all boundaries (see Fig. 8b). The time it may suspend sensing then depends on the shortest distance to qb_O:

$$t_{\text{leave}}(p) = (\text{dist}(p, qb_O) - \delta_{\max})/v_{\max}. \quad (3)$$

The MO must report "leave" if the sensed position is *outside* rb_O, and must not report "leave" while it is *inside* rb_I. The may-report zone remains the same. The resulting algorithm for object-side processing is depicted in Fig. 9.

Note that this algorithm can only provide results within the given tolerance, if the three cases above do not overlap. That is, if rb_I is located completely inside rb_O. Consequently, providing correct query results within the given tolerance requires the following **minimal tolerance for s-CRQ**:

$$\text{dist}(qb_I, qb_O) \geq 2 \cdot \delta_{\max}. \quad (4)$$

```
on obtaining position(pos) do:
       /* check reporting condition */
(1)    if (lastReport ≠ "enter" AND enter-Condition(pos)) then {
(2)          sendReport(id, pos, "enter"); lastReport := "enter";
       } else
(3)    if (lastReport = "enter" AND leave-Condition(pos)) then {
(4)          sendReport(id, pos, "leave"); lastReport := "leave";
       }
       /* low-power mode: */
(5)    if (lastReport = "enter") then {
(6)          schedule next sensing operation after $t_{\text{leave}}$(pos);
(7)    } else {
(8)          schedule next sensing operation after $t_{\text{enter}}$(pos);
       }
enter-Condition(pos):
       /* 1) check must-report case:     */
(9)    if (pos inside  $rb_I$) return true;
       /* 2) check must-not-report case: */
(10)   if (pos outside $rb_O$) return false;
       /* 3) else: may-report case:      */
(11)   return enter-Policy(pos);          // see Section 4.4
leave-Condition(pos):
       /* 1) check must-report case:     */
(12)   if (pos outside $rb_O$) return true;
       /* 2) check must-not-report case: */
(13)   if (pos inside  $rb_I$) return false;
       /* 3) else: may-report case:      */
(14)   return leave-Policy(pos);          // see Section 4.4
```

Fig. 9. Algorithm for processing a tolerant CRQ.

This defines a lower bound on the distance between both query boundaries. With a smaller spatial tolerance, an MO could sense a position with a distance smaller than $\delta_{\max}$ to both query boundaries alike. In that case, the semantics of s-CRQ could no longer be guaranteed.

To ensure correct query semantics, our assumptions regarding the system parameters (τ_{sense}, δ_{sense}, τ_{com}, and $v_{\max}$) must also be fulfilled. In practice, these might be statistical bounds that hold with high probability [17]. For networks with statistical delay bounds, statistical guarantees for satisfying query semantics can thus be provided.

4.3 Spatiotemporal Tolerance

Next, let us consider an st-CRQ, where only MOs continuously located inside qb_{I} in proximity window $<t_{\text{I}}>$ must be included in the result, whereas only MOs continuously located outside qb_{O} during $<t_{\text{O}}>$ must be excluded. For st-CRQ, also the algorithm shown in Fig. 9 can be used. Only the definitions of t_{enter}, t_{leave}, rb_{I}, and rb_{O} have to be adjusted to incorporate the temporal tolerance.

Again, let us consider the "enter" case first. This time, the LM must receive an enter report no later than t_{I} after the MO entered qb_{I} in reality. After sensing its position, say p, an MO can thus suspend sensing for a period

$$t_{\text{enter}}(p) = (\text{dist}(p, qb_{\text{I}}) - \delta_{\max})/v_{\max} + t_{\text{I}}. \tag{5}$$

In addition, the reporting boundaries are adjusted accordingly. Recall that an MO *must* report "enter," if p is inside rb_{I}. As shown in Fig. 10, rb_{I} is therefore located at a distance d_{I} outside qb_{I}, with

$$d_{\text{I}} = \max(\delta_{\text{sense}}, \delta_{\max} - t_{\text{I}} \cdot v_{\max}). \tag{6}$$

To explain this definition, we have to consider two cases:

1. If p is inside qb_{I} or $\text{dist}(p, qb_{\text{I}}) < \delta_{\text{sense}}$, the MO's true position is possibly inside qb_{I} already. In that case, the MO cannot determine precisely when it crossed qb_{I}. Due to the previous suspension time (t_{enter}), it may already be inside qb_{I} for a period $t_{\text{I}} - \tau_{\text{com}}$. Thus, it must send a report immediately, i.e., this causes the first subcondition $d_{\text{I}} \geq \delta_{\text{sense}}$.
2. If p is outside qb_{I} and $\text{dist}(p, qb_{\text{I}}) \geq \delta_{\text{sense}}$, the MO's true position is definitely outside qb_{I}. In that case, the MO can schedule another sensing operation without reporting only if $t_{\text{enter}}(p) \geq 0$. This is represented by the second subcondition $d_{\text{I}} \geq \delta_{\max} - t_{\text{I}} \cdot v_{\max}$, which becomes relevant for small values of t_{I} (i.e., $t_{\text{I}} < \tau_{\text{sense}} + \tau_{\text{com}}$) as shown in Fig. 10.

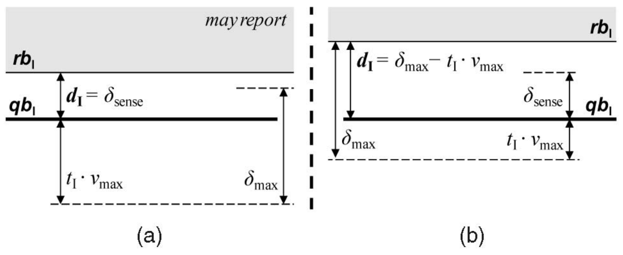


Fig. 10. Reporting boundary for an st-CRQ. (a) $t_{\text{I}} \geq \tau_{\text{sense}} + \tau_{\text{com}}$. (b) $t_{\text{I}} < \tau_{\text{sense}} + \tau_{\text{com}}$.

For "leave" events, the parameters can be determined analogously. An MO that is not included in the result set can schedule the following sensing operation after a time

$$t_{\text{leave}}(p) = (\text{dist}(p, qb_{\text{O}}) - \delta_{\max})/v_{\max} + t_{\text{O}}. \tag{7}$$

It must report "leave" immediately, if the sensed position is outside the reporting boundary rb_{O}, which is located at a distance d_{O} inside qb_{O}, with

$$d_{\text{O}} = \max(\delta_{\text{sense}}, \delta_{\max} - t_{\text{O}} \cdot v_{\max}). \tag{8}$$

As described before (cf. Section 4.2), an MO outside rb_{O} must not report "enter," while an MO located inside rb_{I} must not report "leave." The may-report zone for both events is again located in between rb_{I} and rb_{O}. To guarantee that query results meet the given tolerance, it is again required that rb_{I} is completely included in rb_{O}. Accordingly, query results that are correct within the given tolerance require the following **minimal tolerance for st-CRQ**:

$$\text{dist}(qb_{\text{I}}, qb_{\text{O}}) \geq d_{\text{I}} + d_{\text{O}}. \tag{9}$$

Resolving the *max*-functions of d_{I} and d_{O} thereby yields four inequalities that must all hold to fulfill the condition above:

$$\begin{aligned}
&\text{dist}(qb_{\text{I}}, qb_{\text{O}}) \geq 2 \cdot \delta_{\text{sense}}, \text{and} \\
&\text{dist}(qb_{\text{I}}, qb_{\text{O}}) \geq \delta_{\text{sense}} + \delta_{\max} - t_{\text{I}} \cdot v_{\max}, \text{ and} \\
&\text{dist}(qb_{\text{I}}, qb_{\text{O}}) \geq \delta_{\text{sense}} + \delta_{\max} - t_{\text{O}} \cdot v_{\max}, \text{ and} \\
&\text{dist}(qb_{\text{I}}, qb_{\text{O}}) \geq 2 \cdot \delta_{\max} - (t_{\text{I}} + t_{\text{O}}) \cdot v_{\max}.
\end{aligned} \tag{10}$$

Due to the first inequality, an st-CRQ always requires a minimal spatial tolerance (for any sensor with $\delta_{\text{sense}} > 0$).

Finally, note that s-CRQ is just a special case of st-CRQ. With both temporal tolerances t_{I} and t_{O} set to 0, an st-CRQ exhibits the same semantics as an s-CRQ with the same spatial tolerance.

4.4 Reporting Policy

Now, let us consider the MO's behavior inside the may-report zone. Whenever an application can accept tolerances

```
enter-Policy(pos):
    return ($t_{leave}$(pos) > $t_{enter}$(pos));
leave-Policy(pos):
    return ($t_{enter}$(pos) > $t_{leave}$(pos));
```

Fig. 11. Reporting policy.

that exceed the minimum defined above, this zone has a nonzero width. An MO inside this zone can decide freely whether to report after a sensing operation or not. However, this decision may significantly affect the MO's energy usage. A suitable *reporting policy* must therefore balance the number of sensing and reporting operations.

Clearly, to minimize communication costs, an MO should defer each report as long as possible. That is, it should never send a report inside the may-report zone, but wait until it crosses the second reporting boundary. Yet, this again requires a lot of sensing, as suspension times decrease successively while the MO approaches the respective reporting boundary (analogically to the discussion in Section 4.1). Moreover, the report will often be inevitable later nonetheless. Only if the MO turns around and changes its direction of movement before reaching the second reporting boundary the report can be avoided.

To reduce the total energy consumption, we employ the following policy, which balances reporting and sensing overhead. Inside the may-report zone, an MO sends a report only when this increases the next suspension time. To do so, it compares both t_{enter} and t_{leave} (as depicted in Fig. 11), since the opposite function determines the sensing schedule after a report is sent. For example, consider the "enter" case, in which the MO schedules sensing operations according to t_{enter}. As long as the MO moves toward qb_{I}, t_{enter} is decreasing. At the same time, t_{leave} is increasing due to the growing distance from qb_{O}. After sending an enter report, t_{leave} will be used to schedule the next sensing operation. Thus, the upcoming suspension time is maximized by sending a report as soon as the sensor returns a position p with $t_{\text{leave}}(p) > t_{\text{enter}}(p)$.

This policy effectively minimizes the MO's energy consumption if the MO continues to move toward rb_{I}, and finally crosses this boundary. In that case, it minimizes the sensing overhead without increasing the communication overhead. In our previous work [7], we have carefully evaluated this trade-off for distance-based tolerances. This has confirmed that the energy consumption only increases if a report is sent later. That is, the additional energy spent on communication in some situations is easily amortized by the reduction of sensing operations. This also holds for spatiotemporal tolerances, as we will show in Section 5.

Finally note that, apart from energy consumption, the reporting policy also affects the accuracy of query results. That is, even though the semantics of tolerant CRQ are met at all times, reports may be generated far away from the original query boundary (*qb*). This may cause a large deviation from the "ideal result." In this regard, a different reporting policy could utilize the given tolerance less aggressively (i.e., trade off some energy savings against better query accuracy). But interestingly, both optimization criteria are in fact achieved simultaneously, if tolerances are defined *symmetrically* at both sides of qb (i.e., $\text{dist}(qb,qb_{\text{I}}) = \text{dist}(qb,qb_{\text{O}})$ and $t_{\text{I}} = t_{\text{O}}$). In that case, our reporting policy advises a report as soon as the MO crossed qb, which corresponds to half of the tolerance space.

```
on obtaining position(pos) do:
    /* check all queries:   */
    newReport = false;
    for each q in queries do {
        if (q.lastReport ≠ "enter" AND q.enter-Condition(pos)) then {
            newReport := true; q.lastReport := "enter";
        } else
        if (q.lastReport = "enter" AND q.leave-Condition(pos)) then {
            newReport := true; q.lastReport := "leave";
        }
    }
    if (newReport) then sendReport(id, pos);

    /* low-power mode:  */
    $t_{susp}$ := ∞;
    for each q in queries do {
        if (q.lastReport = "enter") then {
            $t_{susp}$ := min($t_{susp}$, q.$t_{leave}$(pos));
        } else {
            $t_{susp}$ := min($t_{susp}$, q.$t_{enter}$(pos));
        }
    }
    schedule next sensing operation after $t_{susp}$;
```

Fig. 12. Object-side processing of multiple tolerant CRQs.

4.5 Multiple Queries

So far, we only considered a single query. In practice, an MO can be involved in multiple CRQs at the same time, though. Therefore, we now extend the algorithm introduced before to handle multiple queries efficiently and briefly discuss some optimizations of this scheme.

In principle, an MO could simply execute the algorithm independently for each active CRQ. However, we can further optimize the energy usage if the algorithm considers the entire set of (locally known) queries and shares sensing and reporting operations between those queries. The extended algorithm is shown in Fig. 12. Here, we assume that the MO maintains a list of all active CRQs in the variable *queries*. For each query, it contains the tolerance parameters as well as a variable *lastReport* to memorize the query's state. Processing the set of active queries then involves the following three steps: 1) whenever a new position is obtained from the sensor, the MO checks for *all* queries if a report should be sent. For that purpose, it evaluates the *enter-* and *leave-Condition* for each query as discussed before (cf. Fig. 9). 2) If at least one query calls for a report, the MO generates a *single* report message to update all relevant queries at once. To ensure small message sizes, it is even sufficient to report the MO's id and position. Given that the LM knows the MO's parameters ($\delta_{\max}$ and $v_{\max}$) and the reporting policy in use, the LM can deduce independently which query results it has to update. 3) Finally, the next sensing operation is scheduled according to the *minimum* of the suspension times (either t_{enter} or t_{leave}) of each query.

While this algorithm always loops over all active CRQs, we can furthermore improve the local processing time by reducing the number of queries to access in steps 1 and 3 of the algorithm. This can be achieved by introducing a spatial index, like a quad- or R-tree [25], to index the boundaries in main memory. It can be used to efficiently select the queries relevant at a certain position, while taking the tolerances into account. For example, assume that an MO's currently and previously sensed positions are recorded in p_{new} and p_{old}, respectively. Then, the index structure can be used to retrieve all queries with a boundary qb_{O} that contains p_{new} or p_{old} (or both). Only for those queries, the MO has to check whether a report is required (step 1). To compute the next sensing suspension time (step 3), the MO can first retrieve the query boundary closest to p_{new}. If t_{S} is the suspension time for that particular query (which includes its temporal tolerance), then the MO has to consider only those queries located within a distance $t_{\mathrm{S}} \cdot v_{\max}$ to p_{new} in order to determine the minimum suspension time of all queries.

Nevertheless, running too many queries in parallel will eventually overload MOs, which often have limited resources. An interesting approach to adjust the number of queries to an MO's capabilities has been proposed in [3]. Each MO is associated with a so-called *resident domain*, which is a spatial region that surrounds the MO's current position. Only CRQs that intersect with this resident domain are propagated to an MO for object-side processing. Consequently, an MO's load can be adapted by modifying the size of its resident domain. This approach is likewise applicable to tolerant CRQs.

5 Evaluation

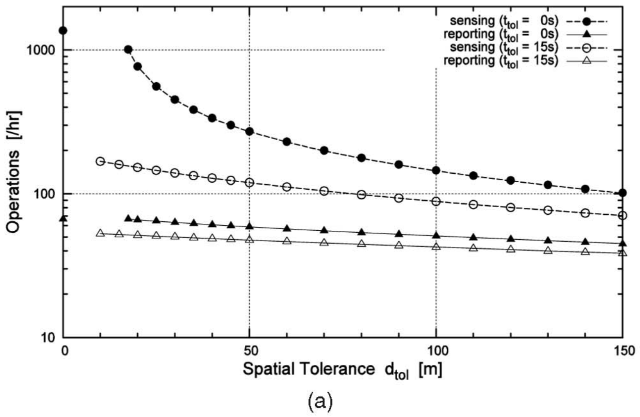
We have performed an extensive experimental evaluation to evaluate the performance of our approaches. In the following, we first explain the simulation setup and then discuss the results in detail.

5.1 Simulation Setup

The experiments are modeled based on a typical scenario of pedestrians commuting inside an European city. It comprises a simulation area of $2.0 \times 2.0\ \mathrm{km}^2$. Movement traces were generated with the CanuMobiSim simulator [27] and follow a smooth motion mobility model [2] through the streets of the city. This model uses stochastic principles to control the change of speed and direction in order to obtain a realistic movement behavior. The target speed is chosen randomly between 0 and 5 m/s every 30 seconds. To simulate typical GPS receivers, a sensing uncertainty is added to each position, which is obtained from a statistical error model based on Gauss-Markow processes [24] with an imprecision ≤ 6.3 m in 95 percent. Each sensing operation takes 0.5 s, and the communication delay is chosen randomly between 0 and 1 s. To assess the impact of these factors of uncertainty, our algorithms use the following upper bounds, respectively: $v_{\max} = 5$ m/s, $\delta_{\mathrm{sense}} = 10$ m, $\tau_{\mathrm{sense}} = 0.5$ s, and $\tau_{\mathrm{com}} = 1$ s.

For each simulation run, we simulated the movements of 100 different MOs over 1 hr. During this time, they had to process a set of 25 CRQs placed randomly within the simulation area. For simplicity, we assume that the boundary of a query is a circle, with radius $r = 200$ m. All presented results were obtained by averaging 100 simulation runs.

For different tolerance sizes, we measured the number of sensing and reporting operations. Then, we used the specification of the Apple iPhone 3GS (a typical smart phone) to determine the lifetime of MOs. It offers a battery capacity of 4.5 Wh (1,219 mAh @3.7 V) and a standby time of 300 hr, which corresponds to a constant background power consumption of 15 mW. Each reporting operation is assumed to cost 150 mJoules to transmit 240 bytes over GSM/GPRS [9]. Each sensing operation consumes 75 mJoules with a low-power GPS receiver [19]. The GPS receiver consumes another 1.5 mW of background power on average.[3]



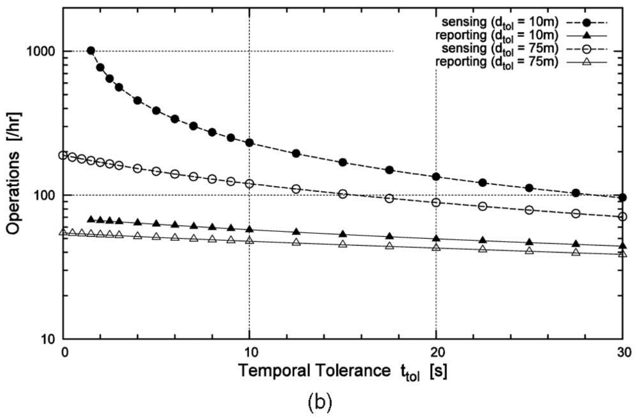


Fig. 13. Sensing and reporting operations performed per hour. (a) d_{tol} and (b) t_{tol}.

We also compare our results with the traditional *object-side processing* technique (e.g., [3], [29]). This is the most efficient solution for CRQs we are aware of (cf. Section 6). However, previous works considered neither the sensing operations nor the different factors of uncertainty. By considering these issues, the methods in [3] and [28] are equivalent to our protocol with zero-tolerance.

5.2 Energy Consumption

First, we evaluated the effect of tolerance values. We incorporated a spatial tolerance to both sides of the query boundaries by increasing and decreasing the radius r by the same distance d_{tol}. Likewise, the same temporal tolerance t_{tol} was added for both MOs entering and leaving a query region (i.e., $t_{\mathrm{I}} = t_{\mathrm{O}} = t_{\mathrm{tol}}$). The values of d_{tol} and t_{tol} are set in a way that satisfy the conditions in (10), i.e., $d_{\mathrm{tol}} \geq 10$ m and $d_{\mathrm{tol}} + t_{\mathrm{tol}} \cdot 5$ m/s ≥ 17.5 m. To assess the energy savings of st-tolerance, we measured the average number of sensing and reporting operations performed by each MO per hour. The result, in logarithmic scale, is depicted in Fig. 13 for

3. Independent of sensing operations, the GPS receiver regularly calibrates its RTC (about 1 sec. on-time every 5 minutes) and updates ephemeris data (approximately 30 sec. on-time every 30 min.) in order to provide fast start-up times (i.e., "hot start" behavior) [17].

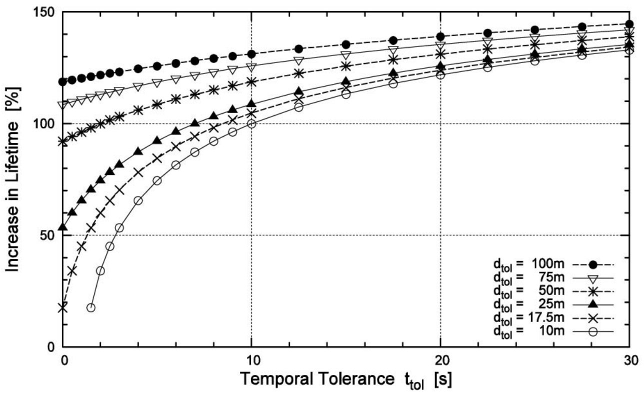


Fig. 14. Increase in lifetime over nontolerant CRQ.

different values of $d_{\rm tol}$ and $t_{\rm tol}$. Fig. 13a also shows the results for zero spatial and temporal tolerances.

We can see that sensing is performed more frequently than reporting, which verifies our approach of focussing primarily on sensing. Without tolerance, an MO has to perform about 20.5 sensing operations per report. However, a minimal tolerance of $d_{\rm tol} = 17.5\ {\rm m}\ (t_{\rm tol} = 0)$ saves 26 percent of sensing operations, while generating the same number of reports. The spatial tolerance prolongs the sensors' suspension times; and very short ones are effectively cut off by sending reports *early*—i.e., before reaching the respective reporting boundary. Larger values of $d_{\rm tol}$ yield even lower sensing cost. For example, $d_{\rm tol} = 35\ {\rm m}$ saves 72 percent of sensing operations over nontolerant processing. The reporting cost also decreases with larger tolerance sizes (7 percent with $d_{\rm tol} = 35\ {\rm m}$). This is because the overlap of adjacent tolerance spaces from different queries increases. Consequently, more query results can be updated simultaneously, with a single report message (cf. Section 4.5).

Fig. 13b shows that increasing the temporal tolerance has a similar effect on sensing and reporting operations. In fact, we can observe that a temporal tolerance t^* yields a close performance as a spatial tolerance $t^* \cdot v_{\max}$. This is because it increases suspension times to the same degree.

Based on these measurements, we also evaluated the impact of tolerance on the overall lifetime of MOs. Without tolerance, an MO's battery is exhausted within 95 hours. The increase in lifetime for different tolerance sizes is depicted in Fig. 14. A minimal tolerance of $d_{\rm tol} = 17.5\ {\rm m}\ (t_{\rm tol} = 0\ {\rm s})$, for example, increases the lifetime by 18.5 percent, while a 10m-10s tolerance yields an increase of 100.5 percent. We can thus conclude that the use of tolerance improves the battery lifetime significantly. Even small tolerances successfully circumvent the critical behavior of object-side processing close to *qb* (cf. Section 4.1). Moreover, this is not achieved at the cost of more report messages. Rather the reporting overhead is also reduced to some extent.

5.3 Accuracy Analysis

Next, we measured the resulting accuracy of query results. Notice that the result of a tolerant CRQ does not deviate from the semantics of query tolerance. Here, we measured the average deviation from the *ideal result*, which is observed in reality without tolerance. In particular, let

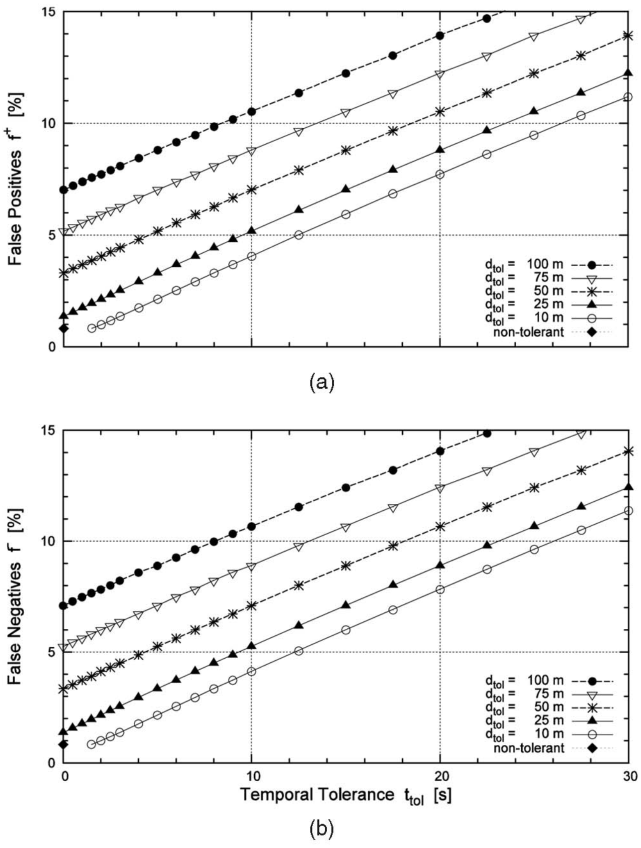


Fig. 15. Deviation from the ideal result. (a) False positives and (b) false negatives.

$RS_{\rm query}(q_i,t)$ be the actual result set, and $RS_{\rm ideal}(q_i,t)$ be the ideal result set of a query q_i at time t. Let $f^+(q_i,t)$ and $f^-(q_i,t)$ denote the fraction of false positives and false negatives, respectively:

$$f^+(q_i,t) := \frac{|RS_{query}(q_i,t) - RS_{ideal}(q_i,t)|}{|RS_{query}(q_i,t)|}, \qquad (11)$$

$$f^-(q_i,t) := \frac{|RS_{ideal}(q_i,t) - RS_{query}(q_i,t)|}{|RS_{ideal}(q_i,t)|}. \qquad (12)$$

Then, we derive the average fraction of false positives f^+ (and of false negatives f^-) over time:

$$f^{+/-} := \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\int_{s_i}^{e_i} f^{+/-}(q_i,t){\rm d}t}{(e_i - s_i)}, \qquad (13)$$

where n denotes the number of active queries, and s_i, e_i denote the start and end times of query q_i, respectively.

Symmetric tolerance. Fig. 15 shows the resulting fraction of false positives and false negatives for the scenario above. Starting at 0.8 percent for minimal tolerance both f^+ and f^- increase linearly with spatial and temporal tolerance. This is due to longer sensor suspension times, which cause an MO to spend more time on the other side of *qb* before acquiring a new position. Nevertheless, the absolute deviation is not very high for moderate tolerance values. For instance, $d_{\rm tol} = 10\ {\rm m}$ and $t_{\rm tol} = 10\ {\rm s}$ result in 4.0 percent (for f^+) and 4.1 percent (for f^-). We also observe that a temporal tolerance t^* again performs like a

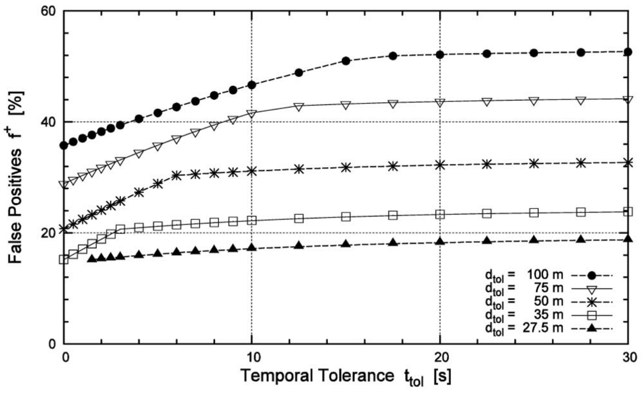


Fig. 16. False positives caused by asymmetric tolerances.

spatial tolerance $t^* \cdot v_{\max}$. As the energy savings were also similar, a temporal tolerance can thus adequately replace a spatial tolerance of that size.

Even more interestingly, we further obtained the same deviation (of $f^+ = f^- = 0.8\%$) for processing the CRQs without tolerance as with minimal tolerances. That is, even though minimal tolerances reduce the energy consumption significantly, they do not affect the accuracy of the query result. The energy is saved only by considering the inevitable technical limitations during algorithm design.

Asymmetric tolerance. Recall from Section 4.4 that symmetric tolerances are most advantageous in terms of query accuracy. However, an application may choose to use different tolerance values inside and outside a query boundary. To assess the effect of "asymmetric tolerance," we repeated the previous experiment with tolerances applied to outside the query boundary only. Specifically, qb_{I} is set to qb, while qb_{O} is obtained by enlarging r by a distance d_{tol}. Likewise, the inner proximity window is zero ($t_{\mathrm{I}} = 0$), and the outer proximity window is varied ($t_{\mathrm{O}} = t_{\mathrm{tol}}$). In this setting, the distance between both boundaries is required to exceed both $\delta_{\max} - \delta_{\mathrm{sense}}$ (i.e., $d_{\mathrm{tol}} \geq 27.5$ m) and $2 \cdot \delta_{\max} - t_{\mathrm{tol}} \cdot v_{\max}$ (i.e., $d_{\mathrm{tol}} + t_{\mathrm{tol}} \cdot 5 \text{ m/s} \geq 35$ m), to satisfy (10).

This does not affect the energy consumption, as it depends solely on the total "width" of the tolerance space. Yet, without inner tolerances st-CRQ prevents any false negatives—a semantic that cannot be achieved with non-tolerant CRQ. On the other hand, the resulting fraction of false positives (depicted in Fig. 16) is larger than in the previous experiment. This is the result of suppressing false negatives. All reports are generated at a "safe distance" outside the query boundary and MOs are thus included in the result for a longer time. With a minimum tolerance, this causes the smallest fraction of false positives feasible without allowing false negatives. With larger tolerance sizes, the fraction of false positives further increases, as MOs utilize the acceptable deviation to reduce power consumption.

We also repeated this experiment with tolerances applied to the inside of qb only and found similar results (with values of f^+ and f^- interchanged). We omit the results here.

5.4 Effect of Query Load

Next, we evaluated the effect of different query loads. For that purpose, we varied the number of active CRQs placed

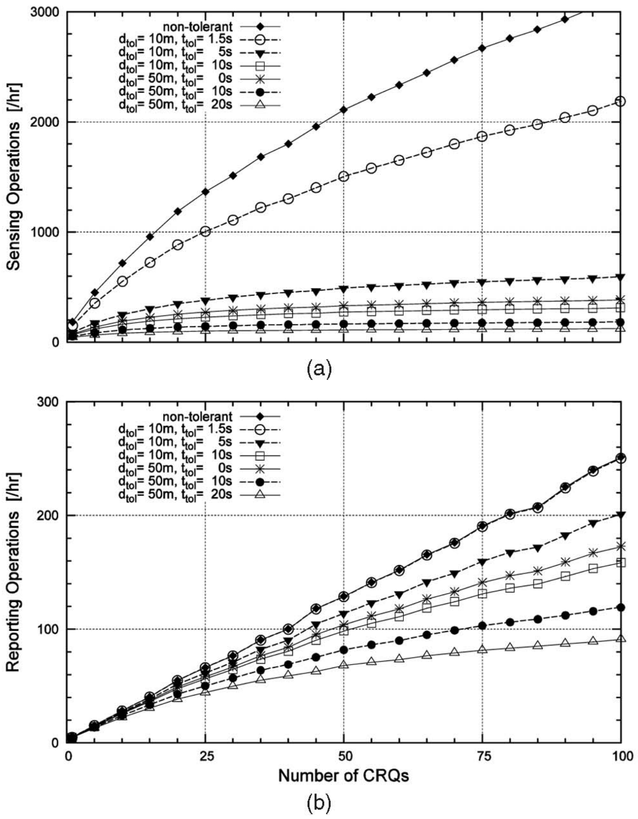


Fig. 17. Increasing the number of active CRQs. The average number of (a) sensing and (b) reporting operations performed by each MO per hour.

randomly within the simulation area, assuming symmetric tolerances again. Fig. 17 depicts the average number of sensing and reporting operations performed by each MO per hour (note the different scales). We observe that the frequency of both operations increases with a growing number of queries. This is because the average distance of an MO to the closest query boundary decreases, so that 1) suspension times become shorter, and 2) more reports are generated. Fig. 17a also reveals that the sensing rate increases slower under larger tolerances. Even with small tolerances, the savings over nontolerant CRQs increase substantially. For example, a minimum tolerance of 10 m and 1.5 s saves 21 percent with five queries, but this increases to 30 percent for 75 queries. With 10 m-10 s tolerance, this changes from 73 to 89 percent. We remark that tolerances provide the highest benefit when an MO is located close to a boundary, which occurs more often with a larger number of CRQs.

Fig. 17b shows that tolerances also save an increasing number of reports. A single query requires the same number of reports regardless of tolerance size. For multiple queries, only minimal tolerances require as many reports as the nontolerant approach. In contrast, a 10 m-10 s tolerance, for example, saves 23.5 percent of reports with 50 queries. As discussed in Section 5.2, this is due to an increasing overlap of tolerance spaces among queries, which allows the updating of more queries with a single message.

In total, the energy savings hence likewise increase with more CRQs to consider. With a 10 m-10 s tolerance, the

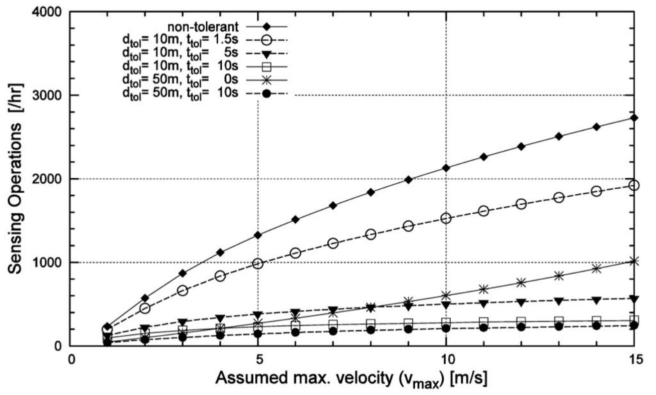


Fig. 18. Increasing the maximum velocity $v_{\max}$.

lifetime of an MO is prolonged by 35 percent when processing five queries, but prolonged by 184 percent with 75 queries.

In another experiment, we used 25 CRQs again and increased the radius r of all query regions. The obtained results are similar: With an increasing size of query regions, an MO crosses a query boundary more often. Thus, the number of reports increases, while suspension times decrease. The advantage of tolerances is also more, and more energy is saved for larger query regions.

In summary, the impact of tolerances increases while an MO is located close to a query boundary. With higher query loads, this occurs more often and thus the energy savings over nontolerant CRQs increase.

5.5 Effect of Movement Speed

Finally, we also studied the effect of faster movement speeds. Note that this actually comprises two different factors of influence. First, MOs become more dynamic when their movement speed increases. Second, this also requires a larger value for $v_{\max}$, the maximum velocity assumed by the algorithm. Interestingly, we found that each of the two factors influences only one of the MOs' operations: the movement speed affects only the number of *reporting* operations, while the number of *sensing* operations is affected only by the assumed $v_{\max}$ parameter. To show this effect, we performed the following experiments:

Sensitivity of $v_{\max}$. First, we increased $v_{\max}$, but left the real movement patterns of MOs unchanged. That is, their speed still ranges from 0 to 5 m/s, but the algorithm uses a less accurate bound of the maximum speed. As a result, the number of reports does not change considerably. Conversely, Fig. 18 shows that generally, the sensing operations increase with $v_{\max}$. This is because a larger $v_{\max}$ lowers the maximal suspension time between sensing operations (cf. Section 4). This effect is cancelled out by a higher temporal tolerance, since it enlarges the suspension times by a constant duration (independent of $v_{\max}$). For example, with a 10 m-10 s tolerance, the number of sensing operations grows just slightly. Spatial tolerances, however, increase the suspension times by a duration of $d_{\mathrm{tol}}/v_{\max}$ only, and thus have a smaller effect for larger $v_{\max}$. We conclude that temporal tolerances are less sensitive to $v_{\max}$.

Note that Fig. 18 also shows values for $v_{\max}$ that are smaller than the true maximum speed of MOs. This clearly yields the lowest sensing costs. However, this gain comes at a price, as the algorithm does no longer provide query results within the given tolerance space. For $v_{\max} = 5$ m/s, Table 2 lists the battery lifetime, the number of reports that arrived at the LM late (i.e., after the MO actually left the tolerance space), and the time those reports are delayed. While the battery lifetime increases with smaller $v_{\max}$, more reports arrive late and the tolerance semantics are no longer provided. In particular, the maximum delay can be very high and hence there is no conceptual bound of the error in query results anymore.

TABLE 2
Decreasing the Maximum Velocity $v_{\max}$

	$v_{\max}$	Delayed reports	Avg. delay	Max. delay	Lifetime
d_{tol} = 10 m,	5.0 m/s	0.00 %	0.0 s	0.0 s	112.4 hr
t_{tol} = 1.5 s	4.0 m/s	0.04 %	0.7 s	4.7 s	121.7 hr
	3.5 m/s	0.6 %	1.3 s	11.8 s	129.1 hr
	3.0 m/s	2.1 %	1.8 s	19.0 s	135.1 hr
	2.5 m/s	7.2 %	3.1 s	46.5 s	146.6 hr
	2.0 m/s	13.8 %	4.4 s	105.1 s	156.5 hr
d_{tol} = 10 m,	5.0 m/s	0.00 %	0.0 s	0.0 s	190.9 hr
t_{tol} = 10 s	4.0 m/s	0.03 %	0.6 s	2.7 s	194.1 hr
	3.5 m/s	0.5 %	1.3 s	10.5 s	196.3 hr
	3.0 m/s	1.6 %	2.0 s	18.4 s	198.6 hr
	2.5 m/s	5.5 %	3.3 s	47.5 s	202.4 hr
	2.0 m/s	10.6 %	4.9 s	83.0 s	205.9 hr
d_{tol} = 50 m,	5.0 m/s	0.00 %	0.0 s	0.0 s	208.5 hr
t_{tol} = 10 s	4.0 m/s	0.08 %	0.6 s	3.3 s	213.1 hr
	3.5 m/s	1.2 %	1.4 s	13.7 s	216,6 hr
	3.0 m/s	3.0 %	2.4 s	18.8 s	219.2 hr
	2.5 m/s	8.4 %	4.6 s	46.2 s	224.0 hr
	2.0 m/s	14.6 %	7.0 s	91.6 s	228.0 hr

If the derivation from the true speed is small (e.g., at 4 m/s), there are only slight penalties in terms of delayed reports, average and maximum delay. This is due to two reasons: First, it is unlikely that MOs move at high in one direction for a long period. Instead, they make turns in between and change the movement speed. Thus, their effective speed toward the query boundary is typically lower than the maximum velocity. Second, the reporting policy does not defer reports until the last possible moment. Reports are typically sent well before reaching the end of the tolerance space.

We conclude that our algorithms are quite robust against small violations of $v_{\max}$ in practice. On the one hand, this is a valuable feature, as MOs may temporarily exceed the assumed maximum velocity without affecting the tolerance semantics significantly. On the other hand, this also indicates the potential for further optimizations by (adaptively) relaxing $v_{\max}$. However, this requires careful consideration, as $v_{\max}$ set too small can cause a large error in query results.

Movement speed. In the next experiment, we tested the effect of varying the MO's movement speed (with $v_{\max}$ fixed at 15 m/s). For this purpose, we generated movement traces as described before, but with a constant speed v. From one experiment to the next, v was gradually increased. This

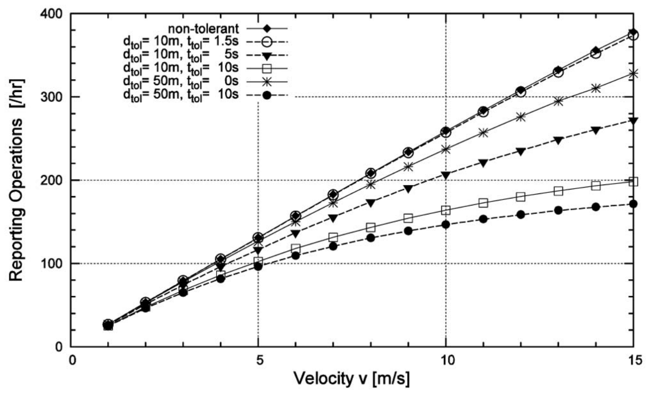


Fig. 19. Increasing actual movement speeds.

time, we found that the number of sensing operations is barely affected. On the other hand, Fig. 19 shows how the number of reporting operations increases, since faster MOs cross query boundaries more frequently. At the same time, the benefits of tolerances also increase with larger v. At $v = 15\ \mathrm{m/s}$, a 10 m-10 s tolerance saves 47.5 percent of reports over nontolerant CRQs.

It follows that faster MOs spend significantly more energy on query processing. First, they have to send reports more often (as shown in Fig. 19). In addition, faster movements also require a larger bound $v_{\max}$, which causes more sensing operations (as shown in Fig. 18). However, we also observe from these figures that the energy savings over the nontolerant solution only increase for faster MOs. With $v = v_{\max} = 15\ \mathrm{m/s}$, for example, the battery lifetime for nontolerant CRQ is 52.5 hours. Yet, the use of minimal tolerances increases the lifetime by 24.5 percent, and a 10 m-10 s tolerance increases it by 187 percent.

5.6 Summary

The concept of st-tolerance provides a novel tool for query users to adjust their accuracy/efficiency trade-off in a flexible way. To end with, we now give a summary of the evaluation results. This can serve as a guideline on how to choose appropriate tolerance parameters in different application domains:

- First, all applications can profit from applying minimal tolerances to both sides of a query boundary (*qb*). As discussed in Section 5.3, this does not affect the accuracy of query results, but it considerably reduces the frequency of sensing operations, and thus the energy consumption.
- Moreover, larger tolerance sizes provide even higher energy savings at the cost of less accurate query results. We obtained significant improvements of an MO's lifetime for only small increases of the tolerance (cf. Section 5.2). Energy savings also increase with higher query loads and faster movement speeds.
- These additional tolerances can be defined in either spatial or temporal domain. A temporal tolerance t^* yields the same performance as the spatial tolerance $t^* \cdot v_{\max}$ (cf. Section 5.3). Yet, the semantics are different. In particular, temporal tolerance provides a tighter bound: An MO can spend an infinite period in spatial proximity to *qb*, but can cover only a limited distance within the temporal proximity window. In addition, spatial tolerance has less impact on MOs with a high maximum velocity $v_{\max}$ (cf. Section 5.5).
- Finally, applying tolerances symmetrically to both sides of *qb* results in the highest accuracy of query results. However, asymmetric tolerances offer important means to limit the amount of either false positives or false negatives (at the cost of increasing the other). Preventing either of them entirely is a unique feature that cannot be achieved without an appropriate definition of tolerance.

6 Related Work

Monitoring the positions of mobile objects has been studied extensively in recent years. In early solutions, all queries are evaluated at the server-side, while MOs are required to constantly report their positions. The characteristics of the employed *reporting protocol* thereby determine the energy consumption of MOs as well as the accuracy of query results [21], [30]. Our previous work has shown how *selective sensing* can be used to reduce the energy consumption of such protocols significantly [6]. While this approach is well suited for continuous tracking of MOs and for spontaneous queries, it is suboptimal for other continuous queries, like CRQs. Since MOs are "query blind," many reports may be sent without affecting any query result.

Many optimizations for continuous queries have been developed recently. But all of them focus on reducing communication costs and do not consider the issues of sensing new position data. First, the idea of "query aware" MOs was introduced by advising MOs of *safe regions*, in which no reports are required [22], [11]. This was carried on by propagating the queries itself to all near-by MOs for *object-side processing*, which results in even lower communication cost. In [3] and [29], this is studied for CRQs, while [18] focuses on continuous nearest-neighbor queries (CNQ), and [9] considers moving CRQs and CNQs. We adopted this approach for our paper.

Conceptually, object-side processing can be classified as a *data stream filtering* technique: The positions of each MO represent a constantly changing data stream; and to evaluate continuous queries over these streams efficiently, a customized *stream filter* is set up at each data source. To improve the performance of such filters, Bobcock and Olston proposed the concept of *tolerance* [1], [20]. This assumes that users can tolerate some degree of imprecision (called *tolerance*) in query results. By incorporating tolerances into filter conditions, they trade off result accuracy for less communication costs. For example, [20] discusses filters for average and minimum queries with value-based tolerance. In [1], filters are designed for top-k queries, using Kalman Filters at each stream is discussed in [12], and filters for nonvalue tolerances are presented in [4].

However, all these works assume that data generated by streams is always correct. They do not consider the different sources of uncertainty introduced by *sensing* data items, as explained in Section 4.1. This can cause stream filters to miss important events, and introduce nontolerated errors. In contrast, we have derived necessary minimum constraints for tolerant CRQs and our algorithms ensure that such tolerances are met despite data uncertainty.

Moreover, none of the aforementioned works considers the energy required for *sensing* new data. They assume that communication is the only relevant factor of energy consumption and thus focus on minimizing the number of report messages. Yet, our results in Section 5 confirm that acquiring new sensor data cannot be ignored, but can instead dominate the energy use of MOs—at least for the prominent sensing technology GPS. In contrast to previous works, we therefore use the given tolerance to maximize sensor suspension times. As a result, an MO can save far more energy than it spends on reporting at all.

On the other hand, *sensing* has already been identified as a relevant source of power consumption in the area of sensor networks [15], [23]. Proposed solutions reduce the energy consumption by *acquiring* new data from a subset of sensors only. Generally, this is achieved by exploiting correlations between values of multiple sensors (either on the same node or on multiple nodes in spatial proximity) and predicting the expected value of others with some level of confidence [13]. However, this differs from the problem considered in this paper, as the positions of different MOs can change independent of each other.

Finally, reducing energy consumption by *selective sensing* has also been studied for activity recognition with body-worn sensors (e.g., [26]). These works focus on the trade-off between energy use and accuracy of recognition, though.

7 Conclusion

In this paper, we addressed the efficient processing of continuous range queries over mobile objects. It has been shown how the concept of *spatiotemporal tolerance* helps to overcome two important challenges: coping with data uncertainty and saving precious energy of MOs.

Limitations of current sensor technologies and communication delay entail different factors of uncertainty that affect the accuracy of query processing. This prevents providing precise results for nontolerant CRQs. In contrast, our definition of spatiotemporally tolerant CRQs offers intuitive, well-defined query semantics in spite of data uncertainty. We derived appropriate minimum constraints for the tolerance size such that our algorithms can compensate all sources of uncertainty and consistently provide results within the given tolerance. Thereby, tolerant CRQs offer applications a lot of flexibility to specify their accuracy requirements precisely. They can specify the acceptable error in both space and time dimensions, independently for both sides of the query boundary. This enables semantics that prevent either false negatives or false positives.

In addition, we have pointed out that *position sensing* operations can have a critically large impact on the energy consumption of MOs. The main reason is that the frequency of these operations has to be increased when an MO is moving closer to the boundary of a nontolerant CRQ. On that account, our algorithms utilize the defined tolerance space effectively to increase sensor suspension times. Our evaluation confirms that this can save a large portion of energy that MOs spend on query processing. Most notably, defining only minimum tolerance sizes symmetrically to both sides of the query boundary reduces the power consumption significantly, without degrading the quality of query results at all. These energy savings further increase, if an application is willing to accept larger tolerances.

In the future, we are going to study how this concept of spatiotemporal tolerance can be applied to other continuous queries (like nearest-neighbor queries, for example).

Acknowledgments

The joint work described in this paper was supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627, by the Germany/Hong Kong Joint Research Scheme (DAAD PPP D/06/00383, G_HK013/06), by the Research Grants Council of Hong Kong (Projects HKU 513307E, HKU 513508E, and HKU 711309E), and by the Seed Funding Programme of the University of Hong Kong (grant no. 200808159002). The authors also thank the reviewers for their insightful comments.

References

[1] B. Babcock and C. Olston, “Distributed Top-K Monitoring,” *Proc. ACM SIGMOD,* June 2003.

[2] C. Bettstetter, “Mobility Modeling in Wireless Networks: Categorization, Smooth Movement, and Border Effects,” *ACM Mobile Computing and Comm. Rev.,* vol. 5, no. 3, pp. 55-66, July 2001.

[3] Y. Cai, K. Hua, G. Cao, and T. Xu, “Real-Time Processing of Range-Monitoring Queries in Heterogeneous Mobile Databases,” *IEEE Trans. Mobile Computing,* vol. 5, no. 7, pp. 931-942, July 2006.

[4] R. Cheng et al., “Adaptive Stream Filters for Entity-Based Queries with Non-Value Tolerance,” *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05),* Sept. 2005.

[5] D. Clark, S. Shenker, and L. Zhang, “Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism,” *ACM Computer Comm. Rev. (SIGCOMM),* vol. 22, no. 4, Oct. 1992.

[6] A. Dey and G. Abowd, “Towards a Better Understanding of Context and Context-Awareness,” Technical Report GIT-GVU-99-22, Georgia Inst. of Technology, College of Computing, 1999.

[7] T. Farrell, R. Cheng, and K. Rothermel, “Energy-Efficient Monitoring of Mobile Objects with Uncertainty-Aware Tolerances,” *Proc. 11th Int'l Database Eng. and Applications Symp. (IDEAS '07),* Sept. 2007.

[8] T. Farrell, R. Lange, and K. Rothermel, “Energy-Efficient Tracking of Mobile Objects with Early Distance-Based Reporting,” *Proc. Fourth Ann. Int'l Conf. Mobile and Ubiquitous Systems (MobiQuitous '07),* Aug. 2007.

[9] B. Gedik and L. Liu, “MobiEyes: A Distributed Location Monitoring Service Using Moving Location Queries,” *IEEE Trans. Mobile Computing,* vol. 5, no. 10, pp. 1384-1402, Oct. 2006.

[10] A. Haeberlen, E. Flannery, A. Ladd, A. Rufys, D. Wallach, and L. Kavraki, “Practical Robust Localization over Large-Scale 802.11 Wireless Networks,” *Proc. ACM MobiCom,* Oct. 2004.

[11] H. Hu, J. Xu, and D.L. Lee, “A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects,” *Proc. ACM SIGMOD,* June 2005.

[12] A. Jain, E. Chang, and Y. Wang, “Adaptive Stream Resource Management Using Kalman Filters,” *Proc. ACM SIGMOD,* June 2004.

[13] Y. Kotidis, “Snapshot Queries: Towards Data-Centric Sensor Networks,” *Proc. 21st Int'l Conf. Data Eng. (ICDE '05),* Apr. 2005.

[14] A. Leonhardi and K. Rothermel, “Architecture of a Large-Scale Location Service,” *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS '02),* July 2002.

[15] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, “The Design of an Acquisitional Query Processor for Sensor Networks,” *Proc. ACM SIGMOD,* June 2003.

[16] P. Misra and P. Enge, *Global Positioning System: Signals, Measurements and Performance,* second ed., Ganga-Jumuna Press, 2006.

[17] S. Moon, J. Kurose, and D. Towsley, "Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms," *ACM/Springer Multimedia Systems,* vol. 6, pp. 17-18, Jan. 1998.

[18] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 11, pp. 1451-1464, Nov. 2005.

[19] Navman, "Jupiter 30 Data Sheet Vers. C," http://www.navmanwirelessoem.com/134.html, May 2007.

[20] C. Olston, J. Jiang, and J. Widom, "Adaptive Filters for Continuous Queries over Distributed Data Streams," *Proc. ACM SIGMOD,* June 2003.

[21] D. Pfoser and C. Jensen, "Capturing the Uncertainty of Moving-Object Representations," *Proc. Sixth Int'l Symp. Spatial Databases (SSD '99),* July 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," *IEEE Trans. Computers,* vol. 51, no. 10, pp. 1124-1140, Oct. 2002.

[23] V. Raghunathan, S. Ganeriwal, and M. Srivastava, "Emerging Techniques for Long Lived Wireless Sensor Networks," *IEEE Comm. Mag.,* vol. 44, no. 4, pp. 108-114, Apr. 2006.

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

[25] H. Samet, "Spatial Data Structures," *Modern Database Systems: The Object Model, Interoperability and Beyond,* W. Kim ed. Addison Wesley/ACM, 1995.

[26] M. Stäger, P. Lukowicz, and G. Tröster, "Power and Accuracy Trade-Offs in Sound-Based Context Recognition Systems," *Pervasive and Mobile Computing,* vol. 3, no. 3, pp. 300-327, June 2007.

[27] I. Stepanov, P. Marron, and K. Rothermel, "Mobility Modeling of Outdoor Scenarios for MANETs," *Proc. 38th Ann. Simulation Symp. (ANSS '05),* Apr. 2005.

[28] U. Varshney, "Location Management for Mobile Commerce Applications in Wireless Internet Environment," *ACM Trans. Internet Technology,* vol. 3, no. 3, pp. 236-255, Aug. 2003.

[29] H. Wang, R. Zimmermann, and W. Ku, "Distributed Continuous Range Query Processing on Moving Objects," *Proc. 17th Int'l Conf. Database and Expert Systems Applications (DEXA '06),* Sept. 2006.

[30] O. Wolfson, A. Sistla, S. Chamberlain, and Y. Yesha, "Updating and Querying Databases that Track Mobile Units," *Distributed and Parallel Databases,* vol. 7, no. 3, pp. 257-387, July 1999.

[31] Zoombak, http://www.zoombak.com, Nov. 2009.

Tobias Farrell received the German Diplom degree in computer science from the University of Stuttgart in 2003. From 2003 to 2009, he was a research assistant and PhD student at the University of Stuttgart. In 2009, he joined Innovations Software Technology GmbH (Bosch Group). His research interests include mobile and context-aware computing, distributed systems, and software engineering.

Kurt Rothermel received the doctoral degree in computer science from the University of Stuttgart in 1985. From 1986 to 1987, he was a postdoctoral fellow at the IBM Almaden Research Center in San Jose and then joined IBM's European Networking Center in Heidelberg, Germany. Since 1990, he has been a professor of computer science at the University of Stuttgart. He is the head of the Institute of Parallel and Distributed Systems and leads the Collaborative Research Center (SFB) Nexus, conducting research in the area of mobile context-aware systems. His research interests are in the fields on distributed systems, computer networks, mobile systems, and sensor networks.

Reynold Cheng received the BEng degree in computer engineering and the MPhil degree in computer science and information systems from the University of Hong Kong (HKU) in 1998 and 2000, respectively, and the MSc and PhD degrees from the Department of Computer Science, Purdue University, in 2003 and 2005, respectively. He is an assistant professor in the Department of Computer Science at HKU. From 2005 to 2008, he was an assistant professor in the Department of Computing at Hong Kong Polytechnic University, where he received two performance awards. He is a member of the IEEE, the ACM, ACM SIGMOD, and UPE. He has served on the program committees and review panels for leading database conferences and journals. He was also a guest editor for the *Transactions on Knowledge and Data Engineering* special issue on mining large uncertain and probabilistic databases in 2010. His research interests include database management, as well as querying and mining of uncertain data.