Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 992–1006
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Application-aware integration of data collection and power management in wireless sensor networks

Qi Han[a,*], Sharad Mehrotra[b], Nalini Venkatasubramanian[b]

[a]*Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, CO 80401, USA*
[b]*Department of Computer Science, University of California, Irvine, CA 92697, USA*

Received 24 March 2006; received in revised form 28 January 2007; accepted 22 May 2007
Available online 31 May 2007

Abstract

Sensors are typically deployed to gather data about the physical world and its artifacts for a variety of purposes that range from environment monitoring, control, to data analysis. Since sensors are resource constrained, often sensor data is collected into sensor databases that reside at (more powerful) servers. A natural tradeoff exists between resources (bandwidth, energy) consumed and the quality of data collected at the server. Blindly transmitting sensor updates at a fixed periodicity to the server results in a suboptimal solution due to the differences in stability of sensor values and due to the varying application needs that impose different quality requirements across sensors. In order to adapt to these variations while at the same time optimizing the energy consumption of sensors, this paper proposes three different models and corresponding data collection protocols. We analyze all three models with a Markov state machine formulation, and either derive closed forms for the operation point of the data collection application or suggest algorithms for estimating this operating point to achieve a minimal energy consumption. We observe that the operating point depends on environmental characteristics and application quality requirements, which the proposed algorithms aim to accommodate. Our experimental results show significant energy savings compared to the naive approach to data collection.
Published by Elsevier Inc.

Keywords: Energy efficiency; Wireless sensor networks; Data collection; Power management

1. Introduction

With the advances in computational, communication, and sensing capabilities, large scale sensor-based distributed environments are becoming a reality. Sensor-enriched communication and information infrastructures have the potential to revolutionize almost every aspect of human life. An integral component of such an infrastructure is a data management system that allows seamless access to data dispersed across a hierarchy of storage, communication, and processing units—from sensor devices where data originates to large databases where the data generated is stored and/or analyzed.

Designing a scalable data management solution to drive distributed sensor applications poses many significant challenges. Given the limited computational, communication, and storage resources at the sensors, a traditional distributed database approach in which sensors function as nodes in a distributed system might not be a feasible option. In order to facilitate complex query processing and analysis, data might need to be migrated to repositories that reside at more powerful servers. An alternative solution, where sensor data is continuously collected at a logically centralized database, might also be infeasible. Since sensor readings may change very frequently and continuously, such environments are highly dynamic. Blindly transmitting the sensor updates to the server will impose severe network and storage overheads. Furthermore, since communication constitutes a major source of power drain [2] in battery-operated sensors, it would incur a very high energy cost.

The problem of effective data collection in highly dynamic environments has recently been studied in [20,24,34]. The key observation is that a large number of sensor applications can tolerate a certain degree of error in data. Data imprecision, of course, impacts application quality. For example, in target tracking using sensor networks, error in sensor intensity

* Corresponding author. Fax: +1 303 273 3875.
E-mail address: qhan@mines.edu (Q. Han).

0743-7315/$ - see front matter Published by Elsevier Inc.
doi:10.1016/j.jpdc.2007.05.008

readings may result in error in localizing the object. Similarly, the result of a query for average temperature in a given region may be imprecise due to data error. The communication overhead between the data producers and the server can be alleviated by exploiting the applications' error tolerance. Motivated by Olston et al. [24], in this paper, we explore data collection protocols for sensor environments that exploit the natural tradeoff between application quality and energy consumption at the sensors. Modern sensors try to be power aware, shutting down components (e.g., radio) when they are not needed in order to conserve energy. We consider a series of sensor models that progressively expose increasing number of power saving states. For each of the sensor models considered, we develop quality-aware data collection protocols that enable quality requirements of the queries to be satisfied while minimizing the resource (energy) consumption.

The rest of this paper is organized as follows. We formulate the quality-aware data collection problem for sensor environments in Section 2. Various sensor models and corresponding data collection protocols to minimize energy consumption are studied in Section 3. Adaptive sensor state management is discussed in Section 4. We analyze the performance in Section 5, discuss related work in Section 6 and conclude in Section 7.

2. Problem formulation

In this section, we describe system and query models used in this paper and develop a formal characterization of the sensor data collection problem.

2.1. System and query model

Our system consists of a set of n sensors and one server (residing at a resource sufficient node) that maintains a database. For simplicity, we will assume that each sensor can communicate directly to the server. Detailed discussion on this assumption can be found at the end of Section 3. Each incoming query Q_i is associated with an accuracy constraint A_i, indicating its tolerance to error in answer precision (we will explain the accuracy constraint and query answer accuracy in the next subsection). Furthermore, a query has a latency bound D which requires that each query be answered within D time units.

Each sensor node has a processor with limited memory, an embedded sensing circuitry, an analog-to-digital converter, and radio circuitry. A micro-operating system controls each component. We only consider different radio modes and assume all the other components are always turned on. We consider three sensor states: active (a), listening (l) and sleeping (s). While the sensor is in the *active* mode, the transmitter and receiver radios are on. While the sensor is in the *listening* state, the receiver radio is on. When a radio is in the idle mode, it is capable of detecting an incoming packet, but not in the process of receiving a packet, so we also classify this mode as the listening state since Tx is off and Rx is on. When the sensor goes to the *sleeping* state, its radio is turned off completely. Even in the sleeping state, changes in sensor values can still be detected, since we assume the sensor and processor are always on.

2.2. Data collection framework

Previous work [3,18,24,25] has shown that an effective approach to exploit the tradeoff between application quality and data imprecision is for the server to maintain an approximate value of the data whose divergence from the true value is guaranteed to be bounded by an error at any time. We next present the data collection framework similar to [24]. Let $S = \{s_1, \ldots, s_n\}$ be the set of sensors. Each sensor hosts its exact value that may change frequently. For each $s_i \in S$, let v_i denote the value stored at sensor s_i. The approximation of v_i is represented by a range r_i with lower bound l_i and upper bound $u_i : r_i = [l_i, u_i]$, which is stored in the database at the server. A query for the value of sensor s_i is answered in the format of a range with a lower and an upper bound, so the answer accuracy is defined by the range size $u_i - l_i$. The accuracy constraint A_i of query Q_i specifies the maximum acceptable width of the result.

Fig. 1 illustrates the data collection process. Whenever the sensor value v_i changes to v_i', sensor s_i checks whether r_i is still a valid approximation for the new value. If v_i falls outside r_i, a new approximation of v_i' is sent to the server to update the database (this process is called *source-initiated update*). Otherwise, there is no need to transmit the update to the server, hence reducing communication overhead. Queries are executed over the cached ranges at the server. If the error tolerance of the query is larger than the data error, i.e., $A_i \geqslant u_i - l_i$, the query is processed without any communication with the sensor. Otherwise, the approximation offered by the database is insufficient, the server may request the exact value from remote sensor. The sensor responds with current exact value and a new approximation to be used by subsequent queries. This process is called *consumer-initiated request and update*.

2.3. Problem statement

Given m user queries, our objective is to minimize sensor energy consumption in the process of answering all m queries. Since a sensor consumes energy even when it is not transmitting or receiving data, besides reducing the communication overhead between a sensor and the server, we also need to minimize the time a sensor is either active or listening even when it is not transmitting updates to the server. Assume that the probabilities of source- and consumer-initiated updates at each time instant are P_{su} and P_{cu}. Formally, we would like to

$$\begin{aligned} \text{minimize} \quad & \bar{E} = E_{\text{su}} \cdot P_{\text{su}} + E_{\text{cu}} \cdot P_{\text{cu}} + E_{\text{extra}} \\ \text{subject to} \quad & (1)\ a_i \leqslant A_i, \quad 0 \leqslant i < m, \\ & (2)\ t_i \leqslant D, \quad 0 \leqslant i < m, \end{aligned}$$

where a_i is the answer accuracy for query i and t_i is the query response time. Also note that, E_{su} is the energy required to send

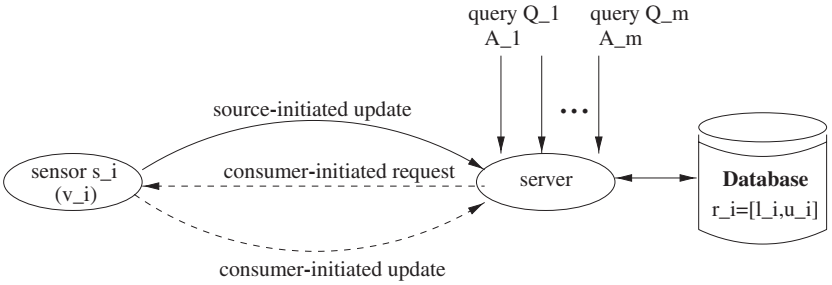

Fig. 1. The data collection process.

an update to the server, and E_{cu} is the energy required to both receive the request for the data and for transmitting the sensor value to the server. Note that E_{su} and E_{cu} are not constant. They depend upon the state at which the sensor was when the source-initiated update and consumer-initiated update occurred. Consider, for example, a sensor that is in the sleeping state. If the sensor value changes and exceeds the range associated for its value at the server, it must first transition to the active state followed by transmitting the value to the server. Thus, the total energy spent would be the sum of energy spent to transition from the sleep state to the active state and the energy spent to transmit the update to the server. In contrast, if the value divergence occurs when the sensor is in the active state, the energy consumption would be only for transmitting the update to the server. E_{extra} is the amount of energy consumed while not receiving or transmitting any data. This is not a constant either, and it depends upon the state a sensor is in while the sensor is free.

To achieve the objective of minimizing the energy consumption at the sensor, we need to address the following issues:

- *How to maintain the database*: An optimal range needs to be maintained and adjusted for each sensor so that it reduces sensor energy consumption while still being able to meet query accuracy constraints. If the range is large, accuracy constraints of many queries will be violated resulting in expensive probes; likewise, if the range is small, sensor update would needlessly be transmitted to the server too frequently. Both cases will consume a large amount of energy. While the time to transmit an update to the server and the time to receive a request from the server is fixed in the collection process, the number of requests and updates is affected by the range size, hence the total energy consumption is dependent upon the choice of range size. We address how to set the range such that the energy consumption is minimized in Section 3.
- *How to manage sensor state*: We need to determine sensor state transition strategies. Sensors consume power not only when sending and receiving data, but also when idling at the active and listening states. To save energy, a sensor needs to power down into a lower energy state. Powering down a sensor requires additional cost to power up when a request that needs to be processed arrives. Furthermore, it could result in increased latency for queries. In Section 4, we address optimal state transition that determines the length of sensor idling and sleeping to minimize overall energy consumption.

3. Energy efficient data precision adjustment

In this section, we deal with how to setup the approximate range (for a sensor) in order to minimize the energy consumption due to communication between sensors and the server. The energy cost due to communication depends upon the number of source- and consumer-initiated updates which, in turn, depends upon the range size adaptation, patterns of the changes in sensor values, and query workload characteristics. Before presenting our solutions, we briefly review the approach described in [24] where the authors consider range adaptation to minimize the communication overhead between data producers and the server. Our approach builds upon some of their results.

Assuming that the communication cost incurred during a source- and consumer-initiated update is C_{su} and C_{cu}, respectively, the expected cost per unit time is $C = P_{su} \cdot C_{su} + P_{cu} \cdot C_{cu}$. The authors established that $P_{su} = \frac{K_1}{r^2}$ and $P_{cu} = K_2 \cdot r$, where r is the range size, and K_1 and K_2 are model parameters that depend on the characteristics of source updates and queries. [1] Therefore, $C = \frac{K_1}{r^2} \cdot C_{su} + K_2 \cdot r \cdot C_{cu}$, which is minimized when the range size $r = \sqrt[3]{\rho \cdot \frac{K_1}{K_2}}$, here $\rho = 2 \cdot \frac{C_{su}}{C_{cu}}$. At this optimal point, it can be shown that the ratio of probability of consumer-initiated updates to the probability of source-initiated updates $\frac{P_{cu}}{P_{su}}$ is equal to ρ, which is a constant.

The algorithm exploits this observation and attempts to change the range r such that the ratio of probability of consumer-initiated update to the probability of source-initiated update can be maintained to be the constant ρ. For example, if $\rho = 1$, the algorithm attempts to ensure that the probability of consumer-initiated updates is equal to the probability of source-initiated updates. In case $\rho < 1$, it is desirable for source-initiated updates to be more frequent than consumer-initiated updates. Thus the range is decreased on every consumer initiated update but only increased with probability ρ

[1] This justification is done under a simplified model, in which the changes in data values follow a random walk model in one dimension, the queries access the data periodically. Although this model is simplified, it is useful for deriving formulas and demonstrating the principles behind our algorithm. As we show empirically in other query workloads and precision constraints that the model is still valid.

```
switch (update-type) {
case source-initiated update:
   with probability min{ρ, 1}, set r' = r(1 + θ); break;
case consumer-initiated update:
   with probability min{1/ρ, 1}, set r' = r/(1+θ); break;
}
return r';
```

Fig. 2. Range size adjustment algorithm proposed in [24].

Table 1
Notations used in the models

Symbol	Meaning
r	Interval size
P_{su}	Probability of source-initiated update at each time instant
P_{cu}	Probability of consumer-initiated update at each time instant
P_i	Probability of a sensor being in state i ($i = a, l, s$)
T_{rx}	Time it takes to receive a consumer-initiated request
T_{tx}	Time it takes to send a source- or consumer-initiated update
T_{ij}	Transition time from state i to j:$i, j = a$(*active*), l(*listening*), s(*sleeping*)
PC_i	Power consumption when sensor is in state i ($i = a, l, s$)
E_{ij}	Energy consumed in switching from state i to j ($i, j = a, l, s$)

on source-initiated updates. Conversely, in case $\rho > 1$, the range is increased on every source-initiated update but only decreased with probability $\frac{1}{\rho}$ on consumer-initiated updates. Fig. 2 explains the algorithm and θ is a parameter that decides how aggressively the range is adjusted. In this case, experiments show that when θ is 1.0 the algorithm performs the best.

We note that the solution in [24] has essentially been developed for data collection in environments where data producers are not energy constrained (e.g., they could be powerful network routers) and its straightforward application is not suitable in energy constrained sensor environments. A direct application of their solution would require that a sensor be always maintained in an active state since a server may need to access the current sensor value at any time which would result in a very high energy cost.

We consider a series of sensor models based on power saving sensor states identified in the previous section. These models progressively consider more sensor states and become more complicated. We start with the always-active (AA) model where sensors are always in the active state. As explained before, this will not perform well in terms of energy saving. It forms a baseline for studying energy savings due to exploiting more sensor states. We then consider the active–listening (AL) model, where sensors switch to the listening state when there are no outstanding requests. The next model considered is the active–sleeping (AS) model. In this model, sensors switch to the sleeping state instead of the listening state if necessary. Finally, we consider the active–listening–sleeping (ALS) model which incorporates an intermediate state (listening) to the AS model. For each of the above models, we discuss the data collection approach that minimizes the energy consumption while meeting the quality constraints of the query. The key issue addressed is how to determine the ranges such that the overall energy consumption is minimized. In deriving the optimal ranges for the various models, we will need some symbols which are summarized in Table 1.

3.1. Always-active (AA) model

In this model, sensors are always active. The total energy consumption normalized over time duration T (i.e., power consumption) is shown as

$$\begin{aligned} \bar{E_{\mathrm{aa}}} &= PC_{\mathrm{a}}(P_{\mathrm{su}}T_{tx}) \quad \text{(source-initiated updates)} \\ &\quad + PC_{\mathrm{a}}(P_{\mathrm{cu}}(T_{rx} + T_{tx})) \quad \text{(consumer-initiated updates)} \\ &\quad + PC_{\mathrm{a}}[1 - P_{\mathrm{su}}T_{tx} - P_{\mathrm{cu}}(T_{rx} + T_{tx})] \quad \text{(idling)} \\ &= PC_{\mathrm{a}}. \end{aligned} \tag{1}$$

As expected, it shows that the normalized energy consumption is equal to the power consumption at the active state. Therefore, irrespective of how the range is set, energy consumption is constant. This model serves as a baseline to study the energy savings that result in utilizing sensor states that consume less energy.

3.2. Active–listening (AL) model

In this model (illustrated in Fig. 3), the sensor consists of two states: active and listening. Initially, the sensor is in the listening mode. The sensor shifts to the active state if either the sensor value diverges from the range used to represent the sensor value at the server, or if it receives a request for its current value from the server. When a sensor is in the active state, it processes all its pending requests and waits for T_{a} time units before switching to the listening mode. We assume T_{a} is a multiple of time unit and $T_{\mathrm{a}} \geqslant 1$. The optimal value of T_{a} that minimizes energy consumption depends upon the application workload and sensor value change patterns. We defer further discussion on how T_{a} can be set in order to minimize power consumption to Section 4. For the time being, we assume that T_{a} has been optimally set. With T_{a} fixed, we consider the problem of optimally determining the range r for the sensor that minimizes the energy consumption.

The sensor energy consumption under this model (Eq. (2)) consists of three parts:

- energy consumed by source-initiated updates. This depends on the sensor state when the source-initiated update is due: if the sensor is listening, there is a power-up energy and also transmission energy;
- energy consumed by consumer-initiated updates. In addition to the energy spent on receiving consumer-initiated requests

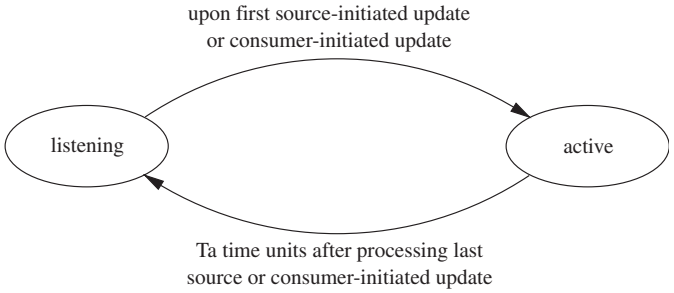

Fig. 3. The active–listening (AL) model.

and transmitting the updates, power-up energy is needed if sensor is listening; and

- energy consumed by the sensor idling in different state. Besides receiving consumer-initiated requests, transmitting source/consumer-initiated updates and transition from listening to active, the sensor also consumes energy by staying in either active or listening state.

$$\begin{aligned}\bar{E_{\mathrm{al}}} = {} & [P_{\mathrm{l}}(E_{\mathrm{la}} + PC_{\mathrm{a}}T_{tx}) + P_{\mathrm{a}}(PC_{\mathrm{a}}T_{tx})]P_{\mathrm{su}} \\ & ((\mathrm{a}) \text{ for source-initiated updates}) \\ & +[P_{\mathrm{l}}(PC_{\mathrm{l}}T_{rx} + E_{\mathrm{la}} + PC_{\mathrm{a}}T_{tx}) \\ & +P_{\mathrm{a}}(PC_{\mathrm{a}}(T_{rx} + T_{tx})]P_{\mathrm{cu}} \\ & ((\mathrm{b}) \text{ for consumer-initiated updates}) \\ & +[1 - (P_{\mathrm{l}}(T_{\mathrm{la}} + T_{tx}) + P_{\mathrm{a}}T_{tx})P_{\mathrm{su}} \\ & -(P_{\mathrm{l}}(T_{rx} + T_{\mathrm{la}} + T_{tx}) + P_{\mathrm{a}}(T_{rx} + T_{tx}))P_{\mathrm{cu}}] \\ & \cdot (PC_{\mathrm{a}}P_{\mathrm{a}} + PC_{\mathrm{l}}P_{\mathrm{l}}) \quad ((\mathrm{c}) \text{ for idling}). \end{aligned} \tag{2}$$

The energy consumption depends upon the probabilities P_{a} and P_{l} of the sensor being in the active and listening state. We next show how these probabilities can be expressed in terms of the probability of source-and consumer-initiated updates. If $T_{\mathrm{a}} = 0$, the sensor state transition matrix capturing the state transition probabilities is as follows:

$$\mathbf{P} = \left(\begin{array}{c|c|c} & \textit{listening} & \textit{active} \\ \hline \textit{listening} & 1 - P_{\mathrm{la}} & P_{\mathrm{la}} = P_{\mathrm{su}} + P_{\mathrm{cu}} \\ \hline \textit{active} & 1 - P_{\mathrm{aa}} & P_{\mathrm{aa}} = P_{\mathrm{su}} + P_{\mathrm{cu}} \end{array} \right). \tag{3}$$

The long-term probability that the system will be in each state can be obtained by computing the steady state vector of the Markov Chain. Therefore, we get $P_{\mathrm{l}} = 1 - P_{\mathrm{su}} - P_{\mathrm{cu}}$ and $P_{\mathrm{a}} = P_{\mathrm{su}} + P_{\mathrm{cu}}$.

As mentioned before, $P_{\mathrm{su}} = \frac{K_1}{r^2}$ and $P_{\mathrm{cu}} = K_2 \cdot r$ [24]. To find the minimum $\bar{E_{\mathrm{al}}}$, we can find the root of the derivative $\frac{d\bar{E_{\mathrm{al}}}}{dr}$, and we get $r^* = \sqrt[3]{\frac{2K_1}{K_2}}$. At this optimal point, $\frac{P_{\mathrm{cu}}}{P_{\mathrm{su}}} = \frac{K_2}{K_1} \cdot r^{*3}$. Thus, energy consumption is minimized when the ratio $\frac{P_{\mathrm{cu}}}{P_{\mathrm{su}}} = 2$ which is a constant.

We next consider the case when $T_{\mathrm{a}} > 0$ (T_{a} is an integer and $T_{\mathrm{a}} \geqslant 1$). In this case the sensor state transition matrix is as follows:

$$\mathbf{P} = \left(\begin{array}{c|c|c} & \textit{listening} & \textit{active} \\ \hline \textit{listening} & 1 - P_{\mathrm{la}} & P_{\mathrm{la}} = P_{\mathrm{su}} + P_{\mathrm{cu}} \\ \hline \textit{active} & P_{\mathrm{al}} & 1 - P_{\mathrm{al}} \end{array} \right), \tag{4}$$

where $P_{\mathrm{al}} = (P_{\mathrm{su}} + P_{\mathrm{cu}}) \left(\left\lceil \frac{1}{T_{\mathrm{a}}(P_{\mathrm{su}} + P_{\mathrm{cu}})} \right\rceil - 1 \right)$ by assuming that source- and consumer-initiated updates are uniformly distributed.[2] Since $x \leqslant \lceil x \rceil < x + 1$, $\lceil x \rceil = x + \alpha, 0 \leqslant \alpha < 1$, we can re-write P_{al} as follows: $P_{\mathrm{al}} = \frac{1}{T_{\mathrm{a}}} - (1 - \alpha)(P_{\mathrm{su}} + P_{\mathrm{cu}}), 0 < \alpha \leqslant 1$. Similar to the calculation of long term state probability when $T_{\mathrm{a}} = 0$, we obtain $P_{\mathrm{l}} = \frac{1 - \alpha T_{\mathrm{a}}(P_{\mathrm{su}} + P_{\mathrm{cu}})}{1 + (1-\alpha)T_{\mathrm{a}}(P_{\mathrm{su}} + P_{\mathrm{cu}})}$ and $P_{\mathrm{a}} = \frac{(P_{\mathrm{su}} + P_{\mathrm{cu}})T_{\mathrm{a}}}{1 + (1-\alpha)T_{\mathrm{a}}(P_{\mathrm{su}} + P_{\mathrm{cu}})}$. We observe that, to minimize $\bar{E_{\mathrm{al}}}$, the optimal $r^* = \sqrt[3]{\frac{2K_1}{K_2}}$. The optimal point occurs when $\frac{P_{\mathrm{cu}}}{P_{\mathrm{su}}} = 2$.

The above analysis shows that for the AL model, independent of the value of T_{a}, the energy consumption is minimized when $\frac{P_{\mathrm{cu}}}{P_{\mathrm{su}}}$ is a constant (equal to 2). Since the data collection protocol proposed in [24] maintains the ratio of the probabilities to be a constant, it can be used in conjunction with the AL model to minimize energy.

3.3. AS model

In the AS model (described in Fig. 4) the sensor toggles between the sleeping and active modes.

Initially, the sensor is in the sleep state. Similar to the AL model, sensor shifts to the active state if the sensor value diverges from the range used to represent the sensor value at the server. Since a sensor in the sleeping state cannot receive requests from the server, it periodically wakes up on a time-out (i.e., if it has been sleeping uninterrupted for T_{s} time units). Such a time-out-based transition is necessary in order to meet the quality requirements of queries that would have resulted in the consumer-initiated update at the sensor. The sensor, on switching to the active state sends its current value to the server. Note that this update can be used by the server to answer those queries that would have resulted in consumer-initiated requests while the sensor was in the sleep state. The sensor remains in the active state while there are requests for its value. After it has handled all the requests, it switches to the sleeping state after waiting for T_{a} time units without handling any requests. We next discuss the energy consumption for the AS model.

In the AS model, total energy consumption (Eq. (5)) consists of the following parts:

- energy consumed by source-initiated updates. Besides the energy spent in transmitting source-initiated updates, there

[2] To explain how P_{al} is derived, let us consider two states i and j. The transition from state i to state j can be caused by either event e_1 with probability P_{e_1} or that the system has been in state i for t time units. To derive state transition probability P_{ij}, two cases are considered: (1) if $1 \left(\mathrm{mod}\, t \cdot P_{\mathrm{e}_1}\right) = 0$, $P_{ij} = \frac{1}{t}$; (2) $P_{ij} = P_{\mathrm{e}_1} + P_{\mathrm{e}_1} \cdot \left\lfloor \frac{1/P_{\mathrm{e}_1}}{t} \right\rfloor$. Combining these two cases, we get $P_{ij} = P_{\mathrm{e}_1} \cdot \left\lceil \frac{1}{t \cdot P_{\mathrm{e}_1}} \right\rceil$.

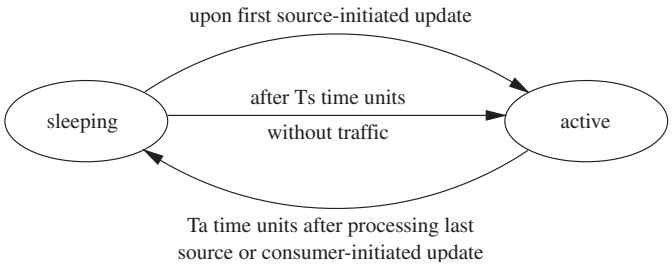

Fig. 4. The active–sleeping (AS) model.

is energy involved in transition from sleeping to active if the sensor is sleeping when the source-initiated update is due;
- energy consumed by transition from the sleeping state to the active state and the associated value updates when a sensor wakes up due to time-out;
- energy consumed by consumer-initiated updates; and
- energy consumed while sleeping or being active without receiving or transmitting.

$$\begin{aligned}\bar{E_{\text{as}}} = {} & [(E_{\text{sa}} + PC_{\text{a}}T_{tx})P_{\text{s}} + PC_{\text{a}}T_{tx}P_{\text{a}}]P_{\text{su}} \\ & \text{((a) for source-initiated updates)} \\ & +(P_{\text{sa}} - P_{\text{su}}P_{\text{s}})(E_{\text{sa}} + PC_{\text{a}}T_{tx}) \\ & \text{((b) for time-triggered updates)} \\ & +[PC_{\text{a}}(T_{rx} + T_{tx})]P_{\text{a}}P_{\text{cu}} \\ & \text{((c) for consumer-initiated updates)} \\ & +(PC_{\text{a}}P_{\text{a}} + PC_{\text{s}}P_{\text{s}})[1 - ((T_{\text{sa}} + T_{tx})P_{\text{s}} + T_{tx}P_{\text{a}})P_{\text{su}} \\ & -(P_{\text{sa}} - P_{\text{su}}P_{\text{s}})(T_{\text{sa}} + T_{tx}) - (T_{rx} + T_{tx})P_{\text{cu}}P_{\text{a}}] \\ & \text{((d) for idling).} \end{aligned} \tag{5}$$

We next derive the optimal setting of the range r for the sensor that minimizes the energy consumption under the assumption that the T_{a} has already been set. As stated before, the optimal setting of T_{a} will be derived in Section 4.

If $T_{\text{a}} = 0$, the sensor switches to the sleeping state as soon as there are no requests waiting. Therefore, $P_{\text{sa}} = P_{\text{su}}\lceil\frac{1}{T_{\text{s}}P_{\text{su}}}\rceil = \frac{1}{T_{\text{s}}} + \alpha P_{\text{su}}$ ($0 \leqslant \alpha < 1$) and $P_{\text{aa}} = P_{\text{su}} + P_{\text{cu}}$. The sensor state transition matrix is as follows:

$$\mathbf{P} = \begin{pmatrix} & \textit{sleeping} & \textit{active} \\ \textit{sleeping} & 1 - P_{\text{sa}} & P_{\text{sa}} \\ \textit{active} & 1 - P_{\text{aa}} & P_{\text{aa}} \end{pmatrix}. \tag{6}$$

Long term probabilities of the sensor being in the sleeping and active states are as follows:

$$P_{\text{s}} = \frac{1 - P_{\text{su}} - P_{\text{cu}}}{\frac{1}{T_{\text{s}}} + \alpha P_{\text{su}} + 1 - P_{\text{su}} - P_{\text{cu}}}, \tag{7}$$

and

$$P_{\text{a}} = \frac{\frac{1}{T_{\text{s}}} + \alpha P_{\text{su}}}{\frac{1}{T_{\text{s}}} + \alpha P_{\text{su}} + 1 - P_{\text{su}} - P_{\text{cu}}}. \tag{8}$$

If $T_{\text{a}} > 0$, the sensor stays active for a period of time so that bursty update requests can be processed without state switching. We can derive state transition probabilities as follows. The probability of switching from sleeping to active:

$$P_{\text{sa}} = P_{\text{su}}\left\lceil\frac{1}{T_{\text{s}}P_{\text{su}}}\right\rceil = \frac{1}{T_{\text{s}}} + \alpha P_{\text{su}}, \quad 0 \leqslant \alpha < 1. \tag{9}$$

The probability of switching from active to sleeping:

$$\begin{aligned} P_{\text{as}} &= (P_{\text{su}} + P_{\text{cu}})\left(\left\lceil\frac{1}{T_{\text{a}}(P_{\text{su}} + P_{\text{cu}})}\right\rceil - 1\right) \\ &= \frac{1}{T_{\text{a}}} - \beta(P_{\text{su}} + P_{\text{cu}}), \quad 0 < \beta \leqslant 1. \end{aligned} \tag{10}$$

The sensor state transition matrix is as follows:

$$\mathbf{P} = \begin{pmatrix} & \textit{sleeping} & \textit{active} \\ \textit{sleeping} & 1 - P_{\text{sa}} & P_{\text{sa}} \\ \textit{active} & P_{\text{as}} & 1 - P_{\text{as}} \end{pmatrix}. \tag{11}$$

Long term state probabilities are

$$P_{\text{s}} = \frac{\frac{1}{T_{\text{a}}} - \beta(P_{\text{su}} + P_{\text{cu}})}{\frac{1}{T_{\text{a}}} - \beta(P_{\text{su}} + P_{\text{cu}}) + \frac{1}{T_{\text{s}}} + \alpha P_{\text{su}}}, \tag{12}$$

and

$$P_{\text{a}} = \frac{\frac{1}{T_{\text{s}}} + \alpha P_{\text{su}}}{\frac{1}{T_{\text{a}}} - \beta(P_{\text{su}} + P_{\text{cu}}) + \frac{1}{T_{\text{s}}} + \alpha P_{\text{su}}}. \tag{13}$$

For both $T_{\text{a}} = 0$ and $T_{\text{a}} > 0$, by applying these probability formula into Eq. (5), the total energy consumption can be expressed as a ratio of two high order polynomials (order of 9) of range size r (see [14] for details). Since it is not possible to express the ratio P_{cu} to P_{su} in terms of other parameters, the basic strategy for range setting described in [24] cannot be used. Instead, we need to monitor parameters K_1, K_2,α and β at runtime. For this purpose, the following information for the sliding window of last k updates is maintained: (1) the number of sensor state transitions (N_{sa} and N_{as}) of the last k updates; and (2) the number of source- or consumer-initiated updates (N_{su} or N_{cu}) of the last k updates. Using this information, the values of K_1, K_2, α and β is estimated. For example, K_1 is set to be $P_{\text{su}} \cdot r^2$, where P_{su} is estimated as the number of source-initiated updates (N_{su}) divided by T, where T is the time period of the current window. The parameter K_2 can be estimated similarly. Given these parameter values, we find the roots of $\frac{d\bar{E_{\text{as}}}}{dr}$ and compare the energy values at the roots to determine the value of r that minimizes the energy consumption $\bar{E_{\text{as}}}$. Since the computation is too complex to be performed at the resource-constrained sensors, it is done at the server. Note that since the value of α and β depends upon the number of sensor state transitions during the window, the sensors determine the values of α and β and piggyback these values for the last k updates with the kth update. At the same time the server monitors K_1 and

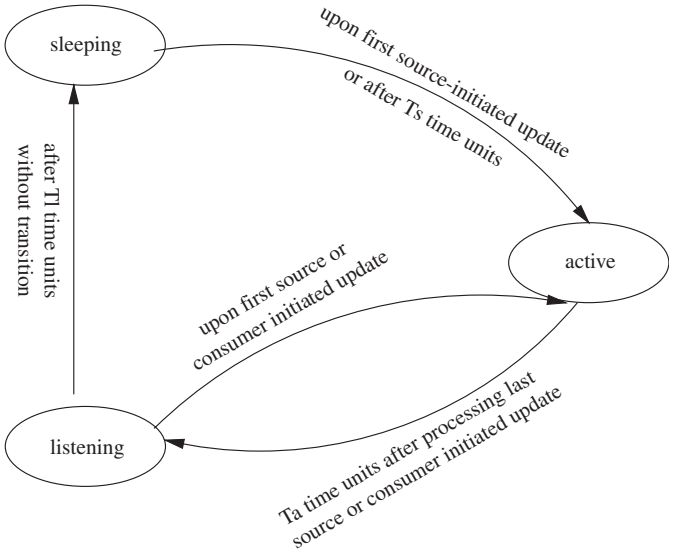

Fig. 5. The active–listening–sleeping (ALS) model.

K_2; upon receiving the kth update, the server computes the new optimal range which is transmitted to the sensor.

3.4. ALS model

In this model (illustrated in Fig. 5), the sensor is initially in the sleeping state. It switches to the active state when a source-initiated update occurs or when it has been sleeping for T_s time units without interruption. When it is in the active state, it processes all the waiting requests. After it has been free in the active state for T_a time units, it goes to the listening state. Once in the listening state, any source- or consumer-initiated update will trigger the sensor to go to the active state; otherwise, if it is idling for T_l time units, it goes to sleep.

The sensor energy consumption (Eq. (14)) consists of the following parts:

- energy consumed by source-initiated updates. Besides the energy spent in transmitting source-initiated updates, there is energy involved in transitions (from sleeping to active if the sensor is sleeping, or from listening to active if the sensor is listening) when the source-initiated update is due;
- energy consumed by transition from the sleeping state to the active state and the associated value updates when a sensor wakes up due to time-out;
- energy consumed by consumer-initiated updates. If the sensor is sleeping, incoming consumer-initiated updates are dropped, so only when sensor is listening or active, would a consumer-initiated request be responded and
- energy consumed while sleeping or being active without receiving or transmitting.

$$
\begin{aligned}
\bar{E_{\mathrm{als}}} = {} & (P_s E_{sa} + P_l E_{la} + PC_a T_{tx}) P_{su} \\
& \text{((a) for source-initiated updates)} \\
& + (P_{sa} - P_{su} P_s)(E_{sa} + PC_a T_{tx}) \\
& \text{((b) for time-triggered updates)} \\
& + [P_l (PC_l T_{rx} + E_{la} + PC_a T_{tx}) \\
& + P_a PC_a (T_{rx} + T_{tx})] P_{cu} \\
& \text{((c) for consumer-initiated updates)} \\
& + (PC_a P_a + PC_l P_l + PC_s P_s) \\
& \cdot [1 - (P_s T_{sa} + P_l T_{la} + T_{tx}) P_{su} \\
& - (P_{sa} - P_{su} P_s)(T_{sa} + T_{tx}) \\
& - ((P_l + P_a)(T_{rx} + T_{tx}) + P_l T_{la}) P_{cu}] \\
& \text{((d) for idling).}
\end{aligned} \tag{14}
$$

Similar to the AS model, we can derive the probabilities of switching between sensor states, from which the steady state probability (P_a, P_l and P_s) can be obtained. Applying these into Eq. (14), the energy consumption can be expressed as a function of r. Similar to the AS model, the optimal range size can be set based on the parameters monitored at runtime. The detailed derivation, though conceptually simple, is quite complex, so we refer the interested readers to [14]. We can apply the same data collection approach as in the AS model.

3.5. Discussion

Our discussion so far assumes a simplistic setting of a single access point (server) that is within one-hop communication range of a set of sensors. In a real system, a server may correspond to an access point that serves as a data collection hub for a set of sensors. Various access points, taken together, may form the overall distributed sensor database. Alternatively, data from these access points may be collected into a centralized sensor data repository. Our solution can serve as a building block for a large scale distributed sensor system.

Previous work [9] has studied the pros and cons of one- and multi-hop communication in wireless sensor networks. The results show that using multi-hop communication is not always optimal in terms of energy consumption. In fact, for typical sensor nodes, the optimum transmission distance to save total energy of transmission and receiving is about 20 m. We believe that there do exist application scenarios such as monitoring of room temperature where the entire network diameter is less than 20 m. Therefore, in those environments, the best communication policy is not to employ multi-hop routing strategies at all; instead, direct transmission from each node to the server is the most energy-efficient communication scheme. Our approaches will work in those environments.

Nevertheless, we do agree that the assumption of one-hop communication limited the applicability of our approaches. To this end, we next discuss how the approaches can be adapted when applied to a two-hop sensor network. A two-hop sensor network can be realized by applying clustering-based communication strategies. For instance, LEACH [16] is such a protocol that groups sensors into enough number of clusters so that each cluster head is one-hop away from the server and each sensor is one-hop away from its cluster head. To balance energy consumption in the network, all the nodes take turns to

be cluster heads. Alternatively, a two-hop sensor network can be realized by using heterogeneous sensing devices and organizing the network into tiers such as in Tenet [13], which has become a widely accepted architecture in the sensor network community. Our data collection protocol and state transition management can be slightly modified to work in a two-hop network. Depending on the role of a sensor node, a different algorithm may be used. For a cluster head as in clustering-based scheme or a master node as in Tenet, we can apply the range size adjustment algorithm proposed in TRAPP [24], the basis of our work. For other nodes, we can apply the data collection protocol presented in the paper.

4. Adaptive sensor state transitions

In the various sensor models discussed above, transitions among states are triggered both by: (a) a sensor value diverging from its representation at the server and (b) time-outs. For example, in the AL and ALS model, a sensor in a sleep mode shifts to the active mode after T_s time units. Furthermore, transition from active/listening to listening/sleeping is also based on time-outs. In this section, we derive how the sensor sets these time-outs in order to maximize energy savings. We believe application layer provides better information about when the radio is not needed, therefore we apply information from upper layers (above the MAC-layer) to control radio power levels.

In the AS and ALS models, a sensor must set a sleeping time T_s after which it transitions to the active state and sends an update to the server. Since sleeping consumes the least energy, it is desirable to maximize the time T_s for which the sensor can sleep. To make sure that all the queries are answered within latency bound D, The worst case query response time for the AS model or the ALS model $T_s + T_{sa} + T_{rx} + T_{tx}$ should not be greater than D. Therefore, T_s is chosen to be $T_s \leqslant D - (T_{sa} + T_{rx} + T_{tx})$.

In AL, AS and ALS model, T_a needs to be determined. We now use the AS model to show how an optimal value of T_a can be derived. The development for the ALS model is similar in nature and interested readers are referred to [14] for details. Waking up a sensor in the sleeping state requires additional energy and latency, so it is not obvious that putting the sensor to sleep immediately after it finishes the requests at hand is the most energy efficient choice. Depending upon the request arrival rate, the sleeping period could be so short that powering up costs are greater than the energy saved in that state. On the other hand, waiting too long to power down may not achieve the best energy reductions possible. Thus, a careful selection of T_a is important. Intuitively, if updates (either initiated by source or consumer) are not bursty, it is better to set T_a to be zero; otherwise, the sensor should remain active for a while before going to sleep, so that more requests can be answered in time and frequent state switching can be avoided. Hence, a good understanding of source- and consumer-initiated update patterns will help in determining the optimal active time. Since the request patterns vary from sensor to sensor, optimal active time T_a also differs from one sensor to another.

We assume that the length of the idle interval can be modeled by a fixed and known distribution whose density function is $\pi(t)$. The expected energy consumption for a single silent period will be

$$E = \int_0^{T_a} \pi(t) P C_a t \, dt + \int_{T_a}^{T_a+T_s} \pi(t)[P C_a T_a + P C_s(t - T_a) + E_{sa}] \, dt. \quad (15)$$

If we assume that requests (either source- or consumer- initiated update requests) are uniformly distributed in interval $(0, T_a + T_s]$ (since we know that at the end of T_s, there must be a time-out update request), then $\pi(t) = \frac{1}{T_a+T_s}$. In this case, the expected energy consumption for a single silent period will be $E = \frac{P C_a T_a^2 + 2 P C_a T_s T_a + (P C_s T_s^2 + 2 E_{sa} T_s)}{2(T_a+T_s)}$. Since $T_a \geqslant 0$ and $E > 0$, E is non-decreasing and is minimal when $T_a = 0$. This implies that the sensor should go to sleep immediately after it finishes all the requests at hand.

While $T_a = 0$ is optimal if request inter-arrival pattern follows uniform distribution, in practice, this assumption is rarely true, and the problem of finding $\pi(t)$ remains. Our approach is to learn $\pi(t)$ at runtime and adaptively select T_a accordingly. The basic idea is as follows: we choose a window size w in advance. The algorithm keeps track of the last w idle period lengths and summarizes this information in a histogram. Every k requests, the histogram is used to generate a new T_a.

The set of all possible inter-arrival period lengths $(0, T_a + T_s)$ is partitioned into n intervals, where n is the number of bins in the histogram. Let t_i be the left endpoint of the ith interval. The ith bin has a counter which indicates the number of idle periods among the last w idle periods whose length fall in the range $[t_i, t_{i+1})$. The bins are numbered from 0 to $n - 1$ and $t_i = 0, t_n = T_a + T_s$.

The counter for bin i is denoted by c_i. The threshold for changing states is selected among n possibilities: $t_0, \ldots, t_{n-1}$. We estimate the distribution π by the distribution which generates an idle period of length t_i with probability c_i/w for $\forall i \in \{0, \ldots, n-1\}$. $\sum_{i=0}^{n-1} c_i = w$. Thus T_a is chosen to be the value t_m that minimizes the following energy consumption:

$$\sum_{i=1}^{m-1} \frac{c_i}{w} P C_a t_i + \sum_{i=m}^{n} \frac{c_i}{w} [P C_a t_m + P C_s(t_i - t_m) + E_{sa}]. \quad (16)$$

Similar derivation can be done to obtain T_a for AL/ALS models and T_l for the ALS model (see details in [14]). Since the frequency and accuracy requirement of user queries vary from sensor to sensor, optimal active time T_a also differs from one sensor to another. Fig. 6 shows the algorithm for computing T_a, which returns the index m and indicates that T_a should be set to be t_m.

Only a sensor itself is aware of its own state transitions, so the above algorithm should be run at each sensor node. Given resource constraints of a sensor, an efficient implementation of the algorithm is necessary. The time complexity of the algorithm every time T_a and T_l need to be updated is $O(n)$ and

```
counter = 0;
E_m = infinity;
Upon a new completed idle period {
    counter ++;
    update histogram;
}
if (counter==k) {
    for (i=0;i<n;i++) {
        compute E_i as in equation 16;
        if E_i < E_m {
            E_m = E_i;
            m= i;
        }
    }
    counter = 0;
    return m;
}
```

Fig. 6. The algorithm for computing T_a.

is independent of the window size w for keeping track of idle period lengths. The frequency with which the active/listening time is updated and the number of bins can significantly affect the time of executing the algorithm. Intuitively, the energy consumption increases as update frequency gets smaller; however, our experiments show that there are no large differences in the cost, we hence update the time for every 50 (i.e., $k = 50$) requests in the performance evaluation. The running time of the algorithm depends linearly on the number of bins, so it is more desirable to select a small value for n. Our experiments show that the results only vary by 1% with n ranging from 1 to 10, we hence set n to be 5.

5. Performance evaluation

The objective of our simulation is to compare the performance of various sensor state models (AA, AL, AS and ALS) for quality-aware data collection in terms of energy consumption and average query response time.

5.1. The simulation environment

The simulated infrastructure consists of a server, a database and a number of sensors (100–500). We assume that the sensors can directly communicate with the server. User queries are posed at the server which then returns their results.

The sensors are characterized as follows. The sensor-related parameters were obtained from the specification of the wireless sensor nodes/motes developed by the University of California-Berkeley [28]. The transmission time (T_{tx}) and reception time (T_{rx}) are derived based on transmission rate 20 kbps of a mote. When a node transmits, it sends a source- or consumer-initiated update which is a 32-bit sensor value; when a node receives, it receives a consumer-initiated request which is a one bit notification. Therefore, we have $T_{tx} = 1.60$ ms and $T_{rx} = 0.05$ ms. Each sensor holds one exact numeric value, and the database holds all the interval approximations. Sensor values are picked randomly and uniformly from the range $[-150, 150]$; they perform a random walk in one dimension: every second, the values either increases or decreases by an amount sampled uniformly from [0.5, 1.5].

User queries arrival times at the server are Poisson distributed with mean inter-arrival time set at 2 s. Each query is accompanied by an accuracy constraint specifying the maximum acceptable width of the result. The accuracy constraints are generated based on parameters $A_{avg} = 20$ (average accuracy constraint) and $A_{var} = 1$ (accuracy constraint variation): they are sampled from a uniform distribution between $A_{min} = A_{avg} \cdot (1 - A_{var})$ and $A_{max} = A_{avg} \cdot (1 + A_{var})$.

5.2. Experimental results

Our performance study first evaluates the proposed sensor models in terms of sensor energy consumption and query response time. We then study the impact of T_s and T_a values, as well as range size adjustment on the system performance. In our system, the two random variables are the changing sensor values and arriving user queries. We analyze the system's behavior by varying the pattern of sensor value changes and query accuracy constraints.

Fig. 7 shows sensor energy consumption and query response time of the four proposed sensor models (AA, AL, AS and ALS). Not surprisingly, the energy consumption of the AA model is the highest, and its query response time is the lowest. This is the model where no energy is saved; sensors are always active, thus any consumer-initiated update requests can be detected immediately and then processed. As shown in Table 2, listening state consumes similar amount of power to active state, thus the AL model does not decrease energy consumption to a great extent. However, most of the time, the sensor is in the listening state when most requests arrive. It switches to the active state so that it can actually send out the updates. This power-up process takes time, which explains why query response time under the AL model is higher than the AA model. Models that incorporate sleeping state reduce energy consumption significantly. However, this comes at the price of higher query response time, since it takes more time for a sensor to switch from the sleeping state to the active state than from the listening state to the active state. Communication cost comparison of these four models show that our energy saving strategies also reduce communication cost slightly. This is because multiple queries waiting at the server can be answered using the database ranges refreshed by the next source-initiated update, avoiding the need for the sensor to transmit an update for each consumer-initiated request. Given that decreasing sensor energy consumption is our objective, the AS model outperforms the other models significantly due to its low energy cost. We, therefore, restrict the remainder of the performance study to the AS model.

Fig. 8 demonstrates the impact of T_s value selection on sensor energy consumption and query response time. When T_s is very small, the sensor powers up from the sleeping state to the active state very often consuming a large amount of energy. However, this benefits queries waiting at the server by shortening their waiting time. As T_s gets larger, the power consumption

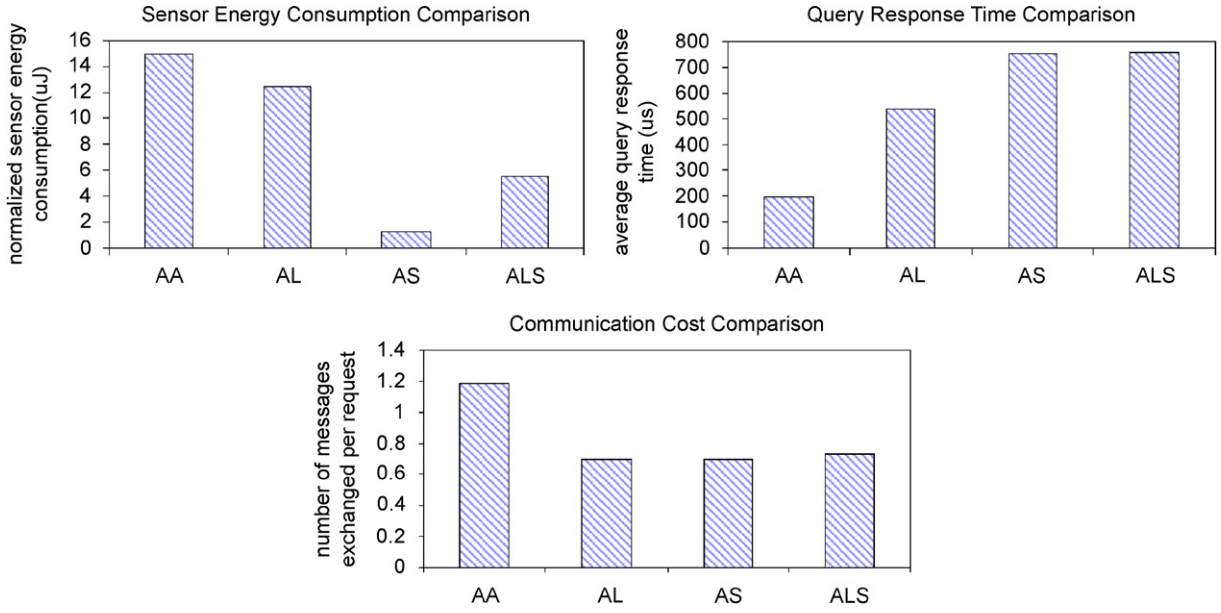

Fig. 7. System performance comparison of proposed sensor models.

Table 2
Parameters used in the simulation

Symbol	Value	Symbol	Value
PC_a	14.88 mW	E_{sl}	0.025 μJ
PC_l	12.50 mW	E_{la}	0.014 μJ
PC_s	0.016 mW	E_{sa}	0.119 μJ
T_{sl}	4 μs	T_{rx}	0.05 ms
T_{la}	12 μs	T_{tx}	1.60 ms
T_{sa}	16 μs		

decreases and query response time increases. After T_s reaches a certain point (100 ms in this case), the energy consumption levels out. This is because the likelihood of sensor switching because of time-out is very low: the energy consumed by source-initiated updates dominates; for the same set of queries and same sensor value change patterns, the energy consumption is similar.

Fig. 9 compares system performance under fixed T_a with system performance using adaptive T_a. Since $T_a = 0$ was shown to be optimal when request arrival follows uniform distribution, we compare the adaptive approach to the approach that fixes T_a to be 0. The results show that adaptive T_a saves energy by half and also decreases query response time. When requests are bursty, it saves energy and shortens query waiting time by remaining active for a certain time period. Adapting T_a to user query patterns and sensor value changes is much better than fixing its value.

Fig. 10 depicts the impact of range size where T_a is adaptive to system conditions. We consider four different cases: (a) $r = 0$: this means the database stores single instantaneous values instead of intervals, (b) set r to be the average accuracy constraint: on an average, queries can be satisfied by stored values, (c) adaptive r as shown in our approaches: the optimal r is found periodically to minimize the energy consumption, and (d) a large r: when r is 0, all queries can be answered by just retrieving values from the database, so query response time is minimized; but each change in sensor value needs to be reported to the server, which consumes a large amount of energy. When r is set to be very large, most source value changes will not exceed current range, so the likelihood of source-initiated updates is low. However, the coarse data representation is not sufficient for most of the queries, hence a number of consumer-initiated updates will occur. As a result, the average query response time is very high. Fig. 10 shows that our adaptive approach significantly outperforms other approaches.

Fig. 11 shows how the system behaves as sensor values change frequency varies where both T_a and range sizes are adaptive to system conditions. When the sensor value changes very frequently, the likelihood of its value falling outside of the current range is high, leading to frequent source-initiated updates. This triggers the sensor to send out updates, and in some cases with the added cost of powering up from sleeping mode; both updates and power-up consume energy. Subsequently, energy consumption is very high. Because the server receives updates constantly, most queries can be answered promptly, and average response time is low. As sensor value changes less frequently, the energy consumption decreases. After it reaches a certain point, the energy consumption evens out. This is because when not enough source-initiated updates occur, the sensor wakes up after T_a time units and sends an update, which dominates energy consumption.

Fig. 12 indicates the impact of query accuracy constraints where both T_a and range sizes are adaptive to system conditions. The horizontal axis is the average accuracy constraint. When most queries need very accurate data, there will be many consumer initiated updates, therefore, both energy consumption and query response time are high. As accuracy constraints

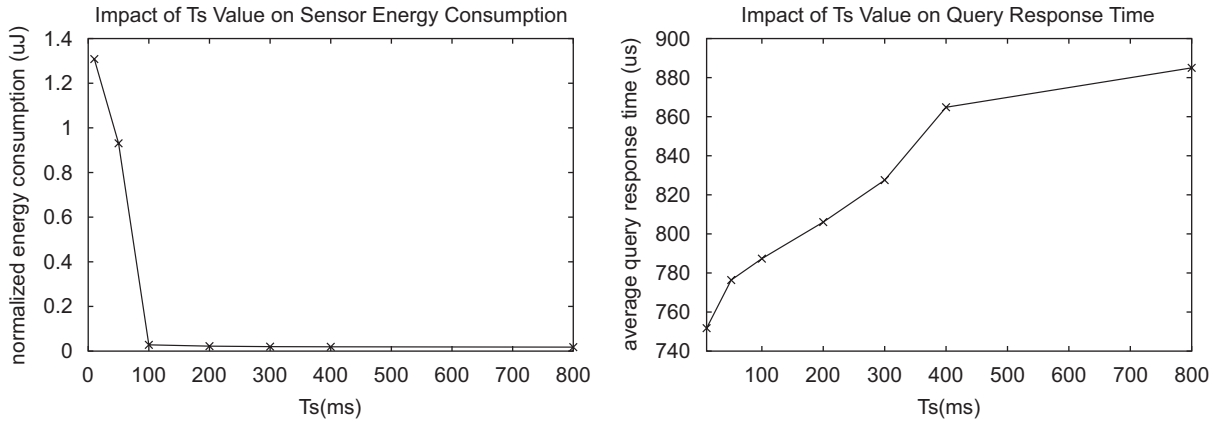

Fig. 8. Impact of T_s value on system performance.

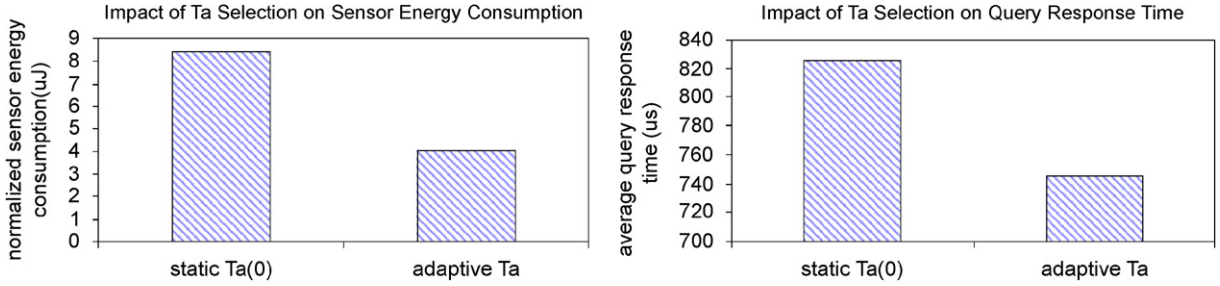

Fig. 9. Impact of T_a adaptation on system performance.

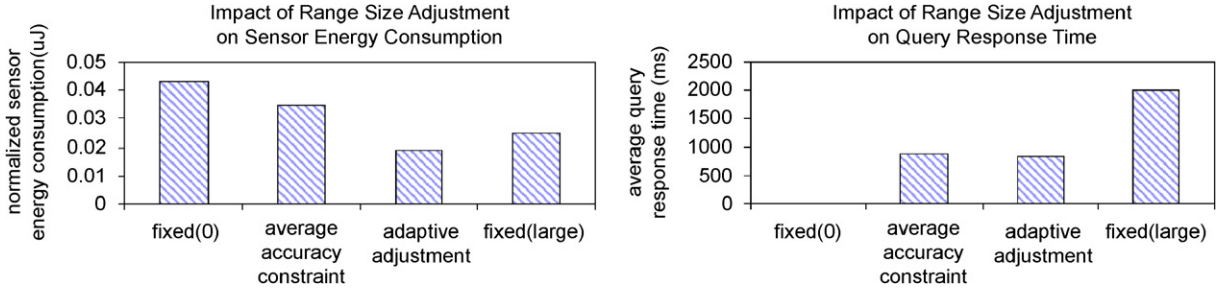

Fig. 10. Impact of range size adaptation on system performance.

are relaxed, increasing number of queries can be answered by just returning the current values in the database, therefore sensor energy consumption becomes smaller and query response time is decreasing. After a certain point, the likelihood of getting consumer-initiated updates becomes very small, and then source-initiated updates and time-out updates consume most of the energy. For this reason, the curve levels off after that point.

5.3. Performance summary

Performance studies indicate that the AS model consumes the least amount of sensor energy; our proposed strategies of intelligent sensor state transition reduce energy consumption to a great extent; optimized range size adjustment works effectively with corresponding sensor models and saves more energy than static range or storing exact values.

6. Related work

In this section, we compare our work to related research in energy efficient protocols for sensor networks, quality aware data collection, sensor databases and data streaming.

6.1. Energy efficient protocols for sensor networks

Energy efficiency is one of the major concerns in sensor networks. To prolong system lifetime of sensor networks, various approaches have been devised to exploit low duty-cycle

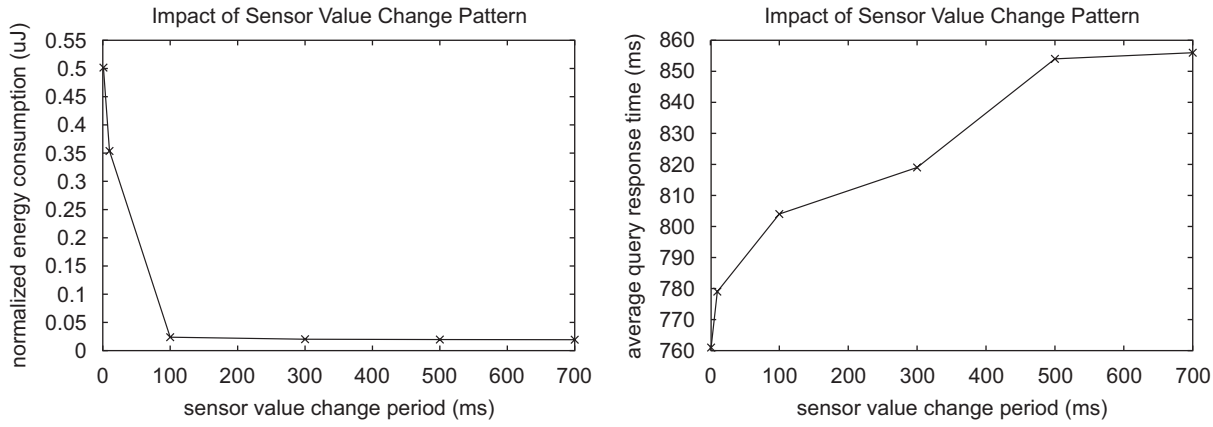

Fig. 11. Impact of sensor value changes on system performance.

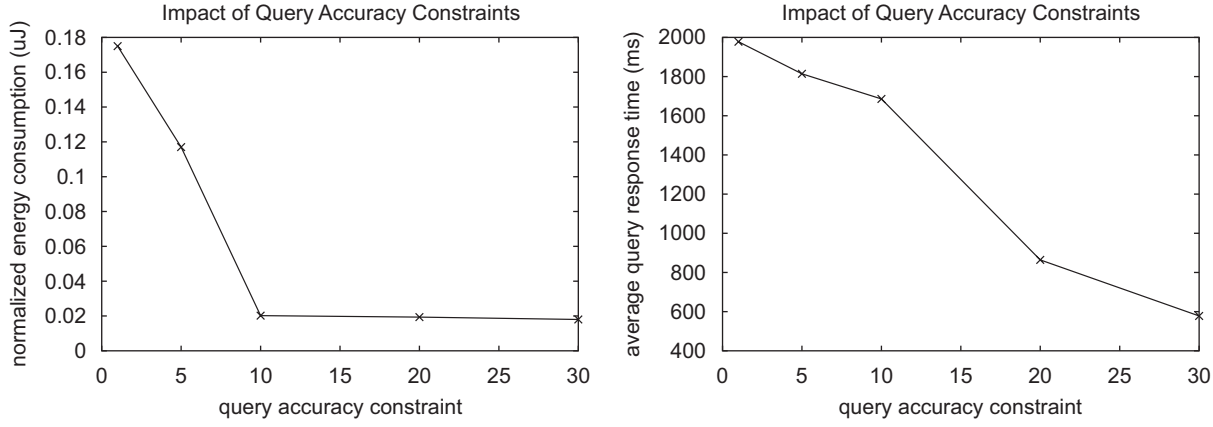

Fig. 12. Impact of query accuracy constraints on system performance.

operation (e.g., transition into a sleeping mode). In ASCENT [8], the decision of when node should go into the sleep state is based on the number of active neighbors and per-link data loss rate. The PEAS [33] protocol avoids the overhead of keeping neighbor state by letting each node probe its surrounding nodes to maintain a desired working node density. STEM [29] trades power savings for path setup latency by using a separate radio operating at a lower duty cycle to detect incoming packets in order to manage sensor network topology. LEACH [16] and PEGASIS [21] aim to balance energy consumption among sensor nodes by rotating cluster heads so that the lifetime of the sensor network can be maximized. More recently, maximizing the lifetime of a two-tiered sensor network is addressed [26], where resource-constrained sensor nodes are deployed around resource-sufficient base stations and at least one application node exists within a sensor cluster. The lifetime is maximized by arranging the locations of base stations and strategically relaying between application nodes. The approaches described above focus on exploiting the cooperation among sensor nodes for better energy conservation. In contrast, our energy saving approaches proposed in this paper take into consideration application-level information. We explore how application's tolerance of information imprecision and data delivery latency can be used to ensure that applications receive the information at their desired levels of accuracy without consuming significant amount of sensor energy. Considering application level information to balance the tradeoff between energy consumption and application quality is also explored in the context of grid [17].

6.2. Quality aware data collection

Balancing the tradeoff between quality and cost has been considered for resource-sufficient information sources, where the major concern is communication overhead. In the context of moving object databases [31], the location of a moving object is predicted based on its velocity vector; decisions of whether or not to update the database is made based on the difference between the cost of information imprecision and the cost of message passing. For more general settings where queries are imposed upon fast changing data, strategies were proposed for answering aggregate queries and maintaining the database at a

reasonable accuracy level [24,25]. Furthermore, quality aware data collection under time constraints is studied [15] where a real-time collection framework, scheduling algorithm and a cost effective database maintenance strategy are proposed and validated. To apply the concept of quality awareness into sensor environments, an optimal online algorithm is proposed for creating piecewise-constant approximation of data to guarantee the quality of sensor data archival applications [20]. This paper aims to support quality awareness for point queries issued on top of dynamically changing sensor data. Point queries differ from data archival applications in that they are associated with implicit or explicit expectation of query response time. In addition, we specifically focused on minimizing energy consumed in the process of answering the queries.

In-network processing has been investigated to balance the tradeoff between energy consumption and quality of aggregate data [12,30]. This paper focuses on exploiting application specified accuracy requirements for more efficient data collection coupled with power management. Similar idea of using application semantics to drive power management has also been explored for real-time sensor applications [11], spatiotemporal queries [5] and rare event detection [7].

6.3. Sensor databases

Research on sensor databases such as COUGAR [6], TinyDB [23], Aurora [1] and Quasar [19] aims to accommodate the special characteristics of sensors into databases. COUGAR describes a data model and long-running query semantics for sensor database systems where stored data are represented as relations and sensor data are represented as time series. The interaction of in-network aggregation with wireless routing protocol for distributed query processing in sensor networks is also investigated in [32]. TinyDB is a query processing system for extracting information from a network of TinyOS sensors. It provides a simple, SQL-like interface to specify the data you want to extract, along with additional parameters, like the rate at which data should be refreshed. Given a query specifying your data interests, TinyDB collects that data from motes in the environment, filters it, aggregates it, and routes it out to a PC. TinyDB does this via power-efficient in-network processing algorithms. Aurora aims to build a single infrastructure that can efficiently and seamlessly meet the various application requirements (e.g., timeliness) in the presence of huge volumes of continuous data streams and data uncertainty. Specifically, it focuses on the real-time data processing issues, such as QoS- and memory-aware operator scheduling, semantic load shedding for coping with transient spikes in incoming data rates, as well as hybrid data storage organizations that would seamlessly and efficiently combine pull- and push-based data processing. Our project Quasar [19] differs from these projects in that it uses the notion of application error tolerance to enhance overall system performance while guaranteeing desired levels of application quality in sensor environments. Our long term goal is to integrate various ideas at the sensor, middleware and application levels into a unified system which will be easily customized to individual sensing applications, but will be generic and modular enough to be useful to a large class of such applications. As part of the ongoing research in Quasar [19], the work presented in this paper explores how application quality tolerance can be used to reduce the energy consumption for query processing in sensor environments. This approach can be viewed as complementing in-network processing. An interesting future work is to combine the in-network query processing with quality-based adaptation to minimize energy consumption.

6.4. Data streaming

Data streaming has been a vibrant research field in recent years (e.g., OpenCQ [22,27], Niagara [10], STREAM [4]). A whole gamut of issues have been addressed, such as system architectures, concepts/semantics of continuous queries, QoS specifications for fresh information delivery, system scalability etc. Our work targets a similar context where data is fast changing and update intensive, but the data generators are resource-limited sensor nodes. This constraint of sensors changed the objective of the problem to be minimization of sensor energy consumption.

7. Conclusions

A key concern of our work is to minimize sensor energy consumption without degrading application quality in distributed sensor environments. In this paper we have shown, using both theoretical analysis and experimental results, how the error tolerance of applications can be exploited to reduce energy consumption while at the same time meeting both accuracy and latency constraints of applications. We have also indicated that, with more complicated sensor models (such as the AS and ALS models), while saving more energy, both sensors and the server have to participate in parameter monitoring and adaptation. Furthermore, we observed that it saves more sensor energy to adaptively determine the duration that the sensor stays in the active state based on source- and consumer-initiated update patterns.

The work presented here focused on collecting data that supports single-item queries. As sensor applications must run in increasingly distributed environments, aggregate queries are often needed. With the same goal of minimizing sensor energy consumption and ensuring application quality, we are in the process of dealing with aggregate continuous queries by taking into account: (1) the spatial and temporal locality of sensor values, and (2) the server's unawareness about sensor states.

In this paper, sensor data is stored in the database in the format of a range with a lower bound and an upper bound, which has been widely explored in caching to reduce communication overhead [24]. More descriptive representation such as probability density functions can be applied to better capture the changes in sensor values. This is part of our ongoing work.

Finally, sensor networks are subject to higher fault rates than traditional networks: connectivity between nodes can be lost due to environmental noise and obstacles; nodes may die due to power depletion, environmental changes or malicious destruction. Providing resilient data collection will require further work that takes into account resource-limited nature of sensor nodes, imprecision of sensor data and the high fault rate of sensor networks.

The eventual goal of our work is to develop effective tools for context information collection in highly dynamic and distributed environments. With the advances of MEMS and wireless networking technologies, sensors are promising to be deployed in a large scale to capture various information from the surroundings. Context-sensitive middleware services will heavily rely on the data obtained from sensors, therefore the work presented in this paper is part of the fundamental support. Middleware techniques for achieving the competing goals of accuracy and energy efficiency such as those described in this paper are key to delivering the right information to the right user at the right time.

References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, Aurora: a new model and architecture for data stream management, VLDB J. 12 (2) (2003) 120–139.

[2] I. Akyildiz, W. Su, Y. Sankarasubramanian, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (2002) 393–422.

[3] G. Apostolopoulos, R. Guerin, S. Kamat, S. Tripathi, Quality of service based routing: a performance perspective, in: ACM SIGCOMM, 1998.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: ACM PODS, 2002.

[5] S. Bhattacharya, G. Xing, C. Lu, G. Roman, B. Harris, O. Chipara, Dynamic wake-up and topology maintenance protocols with spatiotemporal guarantees, in: Proc. of ACM Sensys, 2005.

[6] P. Bonnet, J.E. Gehrke, P. Seshadri, Towards sensor database systems, in: MDM, 2001.

[7] Q. Cao, T. Abdelzaher, T. He, J. Stankovic, Towards optimal sleep scheduling in sensor networks for rare-event detection, in: Proc. of ACM Sensys, 2005.

[8] A. Cerpa, D. Estrin, Ascent: adaptive self-configuring sensor networks topologies, in: InfoCom, 2002.

[9] A. Chandrakasan, R. Min, M. Bhardwaj, S. Cho, A. Wang, Power aware wireless microsensor systems, in: ESSCIRC, 2002.

[10] J. Chen, D.J. DeWitt, F. Tian, Y. Wang, NiagaraCQ: a scalable continuous query system for internet databases, in: SIGMOD, 2000.

[11] O. Chipara, C. Lu, G. Roman, Efficient power management based on application timing semantics for wireless sensor networks, in: IEEE ICDCS, 2005.

[12] A. Deligiannakis, Y. Kotdis, N. Roussopoulos, Hierarchical in-network data aggregation with quality guarantees, in: Proc. EDBT, 2004.

[13] O. Gnawali, B. Greenstein, K. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, E. Kohler, The tenet architecture for tiered sensor networks, in: Proc. of ACM Sensys, 2006.

[14] Q. Han, S. Mehrotra, N. Venkatasubramanian, Energy efficient data collection in distributed sensor environments, Technical Report, University of California-Irvine, 2003.

[15] Q. Han, N. Venkatasubramanian, Addressing timeliness/accuracy/cost tradeoffs in information collection for dynamic environments, in: Proc. of IEEE RTSS, 2003.

[16] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in: HICSS, 2000.

[17] V. Hingne, A. Joshi, E. Houstis, J. Michopoulos, On the grid and sensor networks, in: Proc. Fourth Internat. Workshop on Grid Computing, 2003.

[18] Y. Huang, R. Sloan, O. Wolfson, Divergence caching in client–server architectures, in: IEEE PDIS, 1994.

[19] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. Kalashnikov, W. Yang, Quasar: quality aware sensing architecture, SIGMOD Record 33 (1) (2004) 26–31.

[20] I. Lazaridis, S. Mehrotra, Capturing sensor generated time series with quality guarantees, in: IEEE ICDE, 2003.

[21] S. Lindsey, C. Raghavendra, K.M. Sivalingam, Data gathering algorithms in sensor networks using energy metrics, IEEE Trans. Parallel Distributed Systems 13 (9) (2002) 924–934.

[22] L. Liu, C. Pu, W. Tang, Continual queries for internet scale event-driven information delivery, IEEE TKDE 11 (4) (1999) 610–628.

[23] S.R. Madden, M.J. Franklin, Fjording the stream: an architecture for queries over streaming sensor data, in: IEEE ICDE, 2002.

[24] C. Olston, B.T. Loo, J. Widom, Adaptive precision setting for cached approximate values, in: ACM SIGMOD, 2001.

[25] C. Olston, J. Widom, Offering a precision-performance tradeoff for aggregation queries over replicated data, in: VLDB, 2000.

[26] J. Pan, Y.T. Hou, L. Cai, Y. Shi, S.X. Shen, Topology control for wireless sensor networks, in: ACM MobiCom, 2003.

[27] C. Pu, L. Liu, Update monitoring: the CQ project, in: WWCA, 1998.

[28] RT Monolithics Inc., ⟨http://www.rfm.com/⟩, ASH Transceiver TR1000 Data Sheet, March 2006.

[29] C. Schurgers, V. Tsiatsis, S. Ganeriwal, M. Srivastava, Topology management for sensor networks: exploiting latency and density, in: ACM MobiHoc, 2002.

[30] M. Sharaf, J. Beaver, A. Labrinidis, P. Chrysanthis, Balancing energy efficiency and quality of aggregate data in sensor networks, VLDB J. 13 (4) (2004) 384–403.

[31] O. Wolfson, S. Chamerlain, S. Dao, L. Jiang, G. Mendez, Cost and imprecision in modeling the position of moving objects, in: IEEE ICDE, 1998.

[32] Y. Yao, J. Gehrke, Query processing for sensor networks, in: CIDR, 2003.

[33] F. Ye, G. Zhong, J. Cheng, S. Lu, L. Zhang, Peas: a robust energy conserving protocol for long-lived sensor networks, in: Proc. IEEE ICDCS, 2003.

[34] X. Yu, K. Niyogi, S. Mehrotra, N. Venkatasubramanian, Adaptive middleware for distributed sensor environments, IEEE DS Online 4 (5) (2003) ⟨http://www.dsonline.computer.org⟩.

Qi Han received the PhD degree in Computer Science from the University of California, Irvine in 2005. She is currently an Assistant Professor in the Department of Mathematical and Computer Sciences, Colorado School of Mines. Her research interests include distributed systems, middleware, mobile and pervasive computing, systems support for sensor applications, and dynamic data management. She is specifically interested in developing adaptive middleware techniques for next generation distributed systems. She is a member of the IEEE and the ACM.

Sharad Mehrotra received the PhD degree in Computer Science from the University of Texas at Austin in 1993. He is currently a full Professor in the Department of Computer Science at the University of California, Irvine. He has previously been on the Faculty of the Computer Science Department at the University of Illinois at Urbana-Champaign (1994–1998) and worked as a scientist at the Matsushita Information Technology Laboratory (1993–1994). He specializes in the areas of database management

and distributed systems and has authored more than 85 research publications in top conferences and journals in various disciplines within these fields. He is the recipient of the US National Science Foundation Career Award, Bill Gear Outstanding Junior Faculty Research award at the University of Illinois at Urbana-Champaign in 1997, and the prestigious ACM SIGMOD best paper award in 2001 for a paper entitled "Locally adaptive dimensionality reduction for indexing large time series databases". He is a member of the ACM and the IEEE.

Nalini Venkatasubramanian received the MS and PhD degrees in Computer Science from the University of Illinois, Urbana-Champaign. She is an Associate Professor in the School of Information and Computer Science, University of California, Irvine. Her research interests include distributed and parallel systems, middleware, mobile environments, multimedia systems/applications and formal reasoning of distributed systems. She is specifically interested in developing safe and flexible middleware technology for highly dynamic environments. She was a member of technical staff at Hewlett-Packard Laboratories in Palo Alto, California, for several years, where she worked on large scale distributed systems and interactive multimedia applications. She has also worked on various database management systems and on programming languages/compilers for high-performance machines. She is a member of the IEEE and ACM.