Energy Efficient Data Collection in Distributed Sensor Environments

Qi Han Sharad Mehrotra Nalini Venkatasubramanian
Department of Computer Science, University of California, Irvine, CA 92697-3425
Email: {qhan, sharad, nalini}@ics.uci.edu

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 a sensor database that resides at (more powerful) servers. A natural tradeoff exists between the sensor 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. This paper proposes adaptive data collection protocols for sensor environments that adjusts to these variations while at the same time optimizing the energy consumption of sensors. Our experimental results show significant energy savings compared to the naive approach to data collection.**

I. Introduction

With the advances in computational, communication, and sensing capabilities, large scale sensor-based distributed environments are becoming a reality. Such distributed sensor environments allow us to continuously monitor and record the state of the physical world which can be used for a variety of purposes. It can be used to gain a better understanding of the physical world - e.g., data from embedded loop sensors on highways can be analyzed to understand the emergent traffic patterns. It can also be used to dynamically optimize the process that drives the physical world – e.g., real-time traffic conditions can be used to control the traffic flow. Sensor enriched communication and information infrastructures have the potential to revolutionize almost every aspect of human life benefiting application domains such as transportation, medicine, surveillance, security, defense, science and engineering. 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 resides at (more powerful) server(s). 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/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 [1] 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 [2], [3], [4]. 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 an application such as target tracking in a sensor network, error in sensor intensity 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 [2], in this paper, we explore data collection protocols for sensor environments that exploits the natural tradeoff between application quality and energy consumption at the sensors. Energy conservation is especially critical in environments where it is difficult or infeasible to replenish power supplies of wireless devices. For example, it may be difficult to replace batteries in sensors used to monitor civil infrastructures such as bridges. Similarly, it may be infeasible to replace batteries in smart dust micro-sensors spread over a geographical region for activity monitoring. 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 II. Various sensor models and corresponding data collection protocols to minimize energy consumption are studied in Section III. Adaptive sensor state management is discussed in Section IV. We analyze the performance in section V and discuss related work and conclude in Section VI.

II. 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.

A. 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. 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. Since access points are assumed to be resource sufficient, the more general setting does not pose significant additional challenges. We, therefore, address the collection problem in a simplistic setting of a single access point (server) attached to a set of sensors. Our solution can serve as a building block for large scale distributed sensor system.

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 later). 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 sensor, 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 radio is on; while the sensor is in the *listening* state, the receiver radio is on; and when the sensor goes to the *sleeping* state, its radio is turned off completely. Two main types of microsensor devices that exist in current market are Berkeley Mica Mote [5], [6] and MIT μAMPS node [7], [8]. Mica mote only has one radio, in other words, it either transmits or receives data; while μAMPS node has two radios, and can transmit and receive data simultaneously. These two types of sensor models can both be represented by our abstract sensor models as in Table I. Note that when a radio is in the idle mode if it is capable of detecting an incoming packet, but not currently in the process of receiving a packet. We classify this mode as the listening state since Tx is off and Rx is on. Also note that even in the sleeping state, the changes in sensor values can still be detected, since we assume the sensor and processor are always on. In this paper, we use μAMPS node (i.e., a sensor node with two radios) as the basis of our discussion. For a sensor node with only one radio, similar analysis can be applied.

radio mode		sensor state
Mica mote	μAMPS node	
Tx on Rx off	Tx on **Rx on**	active (a)
Tx off Rx on	Tx off Rx on	listening (l)
Tx off Rx off	Tx off Rx off	sleeping (s)

TABLE I
SENSOR STATES

B. Data Collection Framework

Previous work 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. Specifically, 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.

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., $A_i \geq u_i - l_i$, it 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*. Fig. 1 illustrates the data collection process and Table II describes the data collection protocol at the sensor and server side.

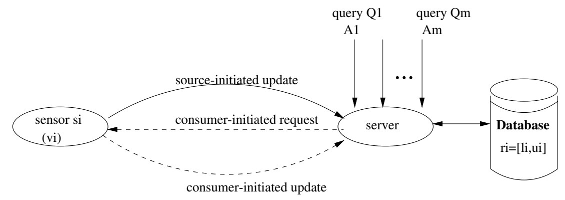


Fig. 1. The Data Collection Process

C. Problem Statement

Given m user queries, our objective is to minimize sensor energy consumption in the process of answering all queries (say m). Since a sensor consumes energy even when it

```
Sensor-Side-Protocol (at time step $n$):
Input Parameters:
    $lastv$ - last value sent to the server;
    $v'[n]$ - current sensor value;
    $\tau$ - timeout value;
    $state$- sensor current state;

$tcounter = 0$;
if (($|v'[n] - lastv| > \frac{r}{2}$) or ($state$==sleeping and $tcounter > \tau$) )
  if ($state$==sleeping)   turn on radio Tx;
  if ($state$==listening)   power up to active state;
  /*apply model-specific processing for source-initiated update*/
  $r' = r$;
  $r'$ =XX-Sensor-Side-Range-Adjustment(sensor-initiated-update, r);
  $l' = v'[n] - \frac{r'}{2}$;$u' = v'[n] + \frac{r'}{2}$;
  send to server: ($l'$, $u'$);
  $lastv = v'[n]$;
  if ($tcounter > \tau$)   $tcounter = 0$;
  else   $tcounter$ ++;
  /* apply model-specific sensor-side monitoring*/
  XX-Sensor-Side-Learning();

if (probe received)
  /*apply model-specific processing for consumer-initiated update*/
  $r'$ =XX-Sensor-Side-Range-Adjustment(consumer-initiated-update, r);
  $l' = v'[n] - \frac{r'}{2}$;$u' = v'[n] + \frac{r'}{2}$;
  send to server: ($l'$, $u'$);
  $lastv = v'[n]$;
```

```
Server-Side-Protocol:
Input Parameters:
    $e$ - query constraint;
    $u$ - current upper bound;
    $l$ - current lower bound;

if (new update ($l'$, $u'$) received)
  $l = l'$; $u = u'$;
  for (each pending consumer-initiated request)
    return answer [$\frac{u-l}{2}$, $\frac{u-l}{2}$];
else
  keep current $l$ and $u$;
  for (each pending consumer-initiated request)
    if ($u - l > e$) /*query constraint not met*/
      probe sensor;
    else
      return answer $[l, u]$;

/*apply model-specific server-side processing*/
XX-Server-Side-Learning();
```

TABLE II

DATA COLLECTION PROTOCOL

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 and/or listening even when it is not transmitting updates to the server. Assume that the probability of source- and consumer-initiated updates at each time instant are P_{su} and P_{cu}. Formally, we would like to

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

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 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 causing it to exceed the range associated for its value at the server, it will first has to 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 also not a constant, and it depends upon the state a sensor is in while it 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 of transmitting an update to the server and the time of receiving 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 III.
- *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 IV, we address optimal state transition that determines the length of sensor idling and sleeping to minimize overall energy consumption.

III. ENERGY EFFICIENT DATA PRECISION ADJUSTMENT

In this section, how the approximation range for the sensor can be set at the server 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 [2] where the authors considered 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 $C = P_{su} \cdot C_{su} + P_{cu} \cdot C_{cu}$. 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. 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 $\frac{P_{cu}}{P_{su}} = \rho$, which is a constant.

The algorithm (explained in Table III) 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 update is equal to the probability of source-initiated update. In case $\rho < 1$, it is desirable for source-initiated updates to be more likely than consumer-initiated updates. Thus the range is decreased on every consumer initiated update but only increased with probability ρ 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.

```
AA-Sensor-Side-Range-Adjustment (update-type, r )
  switch (update-type)
  {
  case source-initiated update:
    with probability $\min\{\rho, 1\}$, set $r' = r(1+\theta)$;
    break;
  case consumer-initiated update:
    with probability $\min\{\frac{1}{\rho}, 1\}$, set $r' = \frac{r}{(1+\theta)}$;
    break;
  }
  return $r'$;
```

TABLE III
RANGE SIZE ADJUSTMENT ALGORITHM PROPOSED IN [2]

We note that the solution in [2] 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.

In the following, we present data collection protocols for sensor environments. 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 Model (AA) 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 Model(AL), where sensors switch to the listening state when there are no outstanding requests. The next model considered is the Active-Sleeping Model(AS). In this model, sensors switch to the sleeping state instead of the listening state if necessary. Finally, we consider the Active-Listening-Sleeping Model(ALS) 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 r_i 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 IV.

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}	the 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 $j(i, j = a, l, s)$
PC_i	power consumption when sensor is in state i $(i = a, l, s)$
E_{ij}	energy consumed in switching from state i to $j(i, j = a, l, s)$

TABLE IV
SYMBOLS USED

A. Always-Active Model (AA)

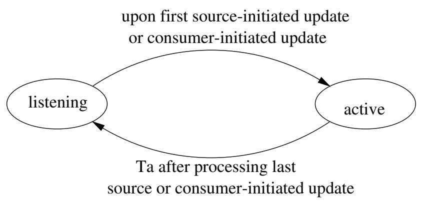
In this model, sensors are always active. The total normalized energy consumption is shown in Equation (1). As

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

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.

B. Active-Listening Model (AL)

In this model(illustrated in Fig. 2), 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 a T_a unit of time before switching to the listening mode. The reason to wait for T_a time units in the active (higher energy) state instead of powering down to the listening (lower energy) state immediately is that switching from a lower energy state to a higher energy state is associated with a significant energy cost. From an energy perspective it might be advantageous to wait in the higher energy state (instead of powering down) if the sensor will be required to transition back to higher energy state in the near future. Obviously, 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 IV. 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



upon first source-initiated update
or consumer-initiated update
listening
active
Ta after processing last
source or consumer-initiated update

Fig. 2. The Active-Listening Model (AL)

sensor energy consumption under this model(Equation (2)) consists of three parts: (a) 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. (b) energy consumed by consumer-intiatied updates. In addition to the energy spent on receiving consumer-initiated requests and transmitting the updates, power-up energy is needed if sensor is listening. (c) energy consumed by the sensor idling in different state. Besides receiving consumer-initiated requests, transmitting source/consumer-initated updates and transitioning from listening to active, the sensor also consumes energy by staying in either active or listening state.

$$\begin{aligned}\bar{E_{al}} &= [P_l(E_{la} + PC_aT_{tx}) + P_a(PC_aT_{tx})]P_{su} \\ &+[P_l(PC_lT_{rx} + E_{la} + PC_aT_{tx}) + P_a(PC_a(T_{rx} + T_{tx})]P_{cu} \\ &+[1 - (P_l(T_{la} + T_{tx}) + P_aT_{tx})P_{su} - (P_l(T_{rx} + T_{la} + T_{tx}) \\ &+P_a(T_{rx} + T_{tx}))P_{cu}](PC_aP_a + PC_lP_l)\end{aligned} \tag{2}$$

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 $T_a = 0$, the sensor state transition matrix capturing the state transition probabilities is as follows:

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

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
$P_l = 1 - P_{su} - P_{cu}$ and $P_a = P_{su} + P_{cu}$.

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

We next consider the case when $T_a > 0$. In this case the sensor state transition matrix is as follows:

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

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

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

C. AS model

In the AS model (described in Figure 3) 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 sleep state cannot receive request from the server, it periodically wakes up on a timeout if it has been sleeping uninterrupted for T_s time units. Such a timeout 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(Equation (3)) consists of (a) energy consumed by source-initiated updates. Besides the energy spent in transmitting source-initiated updates, there is energy involved in transitioning from sleeping to active if the sensor is sleeping when the source-initiated update

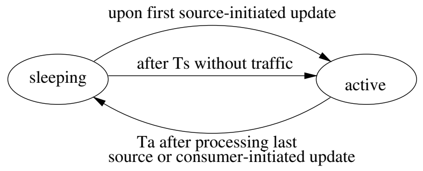

upon first source-initiated update
after Ts without traffic
sleeping
active
Ta after processing last source or consumer-initiated update

Fig. 3. The Active-Sleeping Model (AS)

$$\begin{aligned}\bar{E_{as}} &= [(E_{sa} + PC_aT_{tx})P_s + PC_aT_{tx}P_a]P_{su} \\ &+(P_{sa} - P_{su}P_s)(E_{sa} + PC_aT_{tx}) \\ &+[PC_a(T_{rx} + T_{tx})]P_aP_{cu} \\ &+(PC_aP_a + PC_sP_s)[1 - ((T_{sa} + T_{tx})P_s + T_{tx}P_a)P_{su} \\ &+(P_{sa} - P_{su}P_s)(T_{sa} + T_{tx}) + (T_{rx} + T_{tx})P_{cu}P_a]\end{aligned} \tag{3}$$

is due; (b) 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; (c)energy consumed by consumer-initiated updates; and (d) energy consumed while sleeping or being active without receiving or transmitting.

We next derive the optimal setting of the range r_i 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 IV.If $T_a = 0$, the sensor switches to the sleeping state as soon as there is no requests waiting. Therefore,
$P_{sa} = P_{su}\lceil\frac{1}{T_sP_{su}}\rceil = \frac{1}{T_s} + \alpha P_{su}, 0 \leq \alpha < 1.$
$P_{aa} = P_{su} + P_{cu}$

The sensor state transition matrix is as follows:

$$\mathbf{P} = \left(\begin{array}{c|c|c} & \textit{sleeping} & \textit{active} \\ \hline \textit{sleeping} & 1-P_{sa} & P_{sa} \\ \hline \textit{active} & 1-P_{aa} & P_{aa}\end{array}\right)$$

Long term probabilities of the sensor being in the sleeping and active states are as follows:
$P_s = \frac{1-P_{su}-P_{cu}}{\frac{1}{T_s}+\alpha P_{su}+1-P_{su}-P_{cu}}$ and $P_a = \frac{\frac{1}{T_s}+\alpha P_{su}}{\frac{1}{T_s}+\alpha P_{su}+1-P_{su}-P_{cu}}$.

If $T_a > 0$, 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:

$$P_{sa} = P_{su}\lceil\frac{1}{T_sP_{su}}\rceil = \frac{1}{T_s} + \alpha P_{su}, 0 \leq \alpha < 1$$

The probability of switching from active to sleeping:

$$\begin{aligned}P_{as} &= (P_{su} + P_{cu})(\lceil\frac{1}{T_a(P_{su} + P_{cu})}\rceil - 1) \\ &= \frac{1}{T_a} - \beta(P_{su} + P_{cu}), 0 < \beta \leq 1\end{aligned}$$

The sensor state transition matrix is as follows:

$$\mathbf{P} = \left(\begin{array}{c|c|c} & \textit{sleeping} & \textit{active} \\ \hline \textit{sleeping} & 1-P_{sa} & P_{sa} \\ \hline \textit{active} & P_{as} & 1-P_{as}\end{array}\right)$$

Long term state probabilities are:
$P_s = \frac{\frac{1}{T_a}-\beta(P_{su}+P_{cu})}{\frac{1}{T_a}-\beta(P_{su}+P_{cu})+\frac{1}{T_s}+\alpha Psu}$ and
$P_a = \frac{\frac{1}{T_s}+\alpha Psu}{\frac{1}{T_a}-\beta(P_{su}+P_{cu})+\frac{1}{T_s}+\alpha P_{su}}$.

For both $T_a = 0$ and $T_a > 0$, by applying these probability formula into Equation 3, the total energy consumption can be expressed as a ratio of two complex polynomials of range size r (see [9] for detail). Since 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 Table III can not 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_a) 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 $P_{su} \cdot r^2$, 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 $\frac{d\bar{E_{as}}}{dr}$ and compare the energy values at the roots to determine the value of r that minimizes the energy consumption $\bar{E_{as}}$. 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 k^{th} update. At the same time the server monitors K_1 and K_2; upon receiving the k^{th} update, the server computes the new optimal range which is transmitted to the sensor. Table V shows the data collection protocol at both server and sensor side.

D. ALS Model

In this model (illustrated in Fig. 4), 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 initiated update 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 (Equation (4)) consists

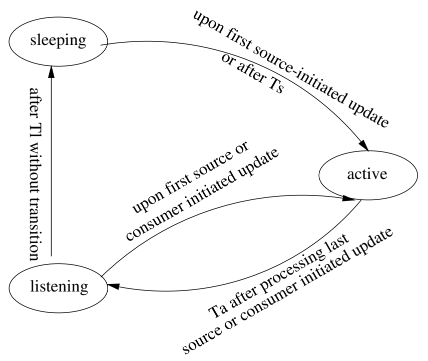

sleeping
upon first source-initiated update or after Ts
active
after Tl without transition
upon first source or consumer initiated update
Ta after processing last source or consumer initiated update
listening

Fig. 4. The Active-Listening-Sleeping Model (ALS)

of (a) energy consumed by source-initiated updates. Besides

```
**AS-Sensor-Side-Learning:**
Variables:
  $N_{as}$ - number of transitions from active to sleeping
  $N_{sa}$ - number of transitions from sleeping to active
  $P_{as}$ - probability of transitions from active to sleeping
  $P_{sa}$ - probability of transitions from sleeping to active

**while** (1)
{
  **if** (transition from active to sleeping)   $N_{as}$ ++;
  **if** (transition from sleeping to active)   $N_{sa}$ ++;
  **if** (received an update) {
    $i++$;
    **if** ($i == k$)   {
      $P_{as} = N_{as}/T$;   $P_{sa} = N_{sa}/T$;
      compute $\alpha$ and $\beta$ for current sliding window;
      send to server: $(\alpha, \beta)$;
      $i = 0$;
    }
  }
}
**AS-Server-Side-Learning:**
Variables:
  $N_{su}$ - number of source-initiated updates;
  $N_{cu}$ - number of consumer-initiated updates;
  $P_{su}$ - probability of source-initiated updates;
  $P_{cu}$ - probability of consumer-initiated updates;

**while** (1)
{
  **if** (received an update)
  {
    i++;
    **if** (it is source-initiated update)   $N_{su}$ ++;
    **else if** (it is consumer-initiated update)   $N_{cu}$ ++;
    **if** ($i == k$) {
      $P_{su} = N_{su}/T$;   $P_{cu} = N_{cu}/T$;
      compute $K_1$ and $K_2$ for current sliding window;
      compute $r$;
      send to sensor: $r$;
    }
  }
}
```

TABLE V

DATA COLLECTION PROTOCOL FOR THE AS MODEL

the energy spent in transmitting source-initiated updates, there is energy involved in transitioning (from sleeping to active if the sensor is sleeping, or transitioning from listening to active if the sensor is listening) when the source-initiated update is due; (b) 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; (c)energy consumed by consumer-initiated updates. If the sensor is sleeping, incoming consumer-intiiated updates are dropped, so only when sensor is listening or active, would a consumer-initiated request be responded. and (d) energy consumed while sleeping or being active without receiving or transmitting.

$$\begin{aligned} \bar{E_{als}} \quad &= (P_s E_{sa} + P_l E_{la} + PC_a T_{tx}) P_{su} \\ &+ (P_{sa} - P_{su} P_s)(E_{sa} + PC_a T_{tx}) \\ &+ [P_l (PC_l T_{rx} + E_{la} + PC_a T_{tx}) + P_a PC_a (T_{rx} + T_{tx})] P_{cu} \\ &+ (PC_a P_a + PC_l P_l + PC_s P_s) \\ &\cdot [1 - (P_s T_{sa} + P_l T_{la} + T_{tx}) P_{su} - (P_{sa} - P_{su} P_s)(T_{sa} + T_{tx}) \\ &- ((P_l + P_a)(T_{rx} + T_{tx}) + P_l T_{la}) P_{cu}] \end{aligned} \tag{4}$$

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 Equation (4), the energy consumption can be expressed as a function of r. Similar to the AS model, 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 [9]. We can apply the same data collection approach as in the AS model shown in Table V.

Discussion: For Mica mote (i.e., a sensor node with only one radio), the sensor goes to the active state only if it has some data to send. While the sensor is in the active state, its Rx radio is turned off and will not detect any incoming consumer-initiated requests. Therefore, the total energy consumption should be modified accordingly to reflect this. For example, in the AS model, all the consumer-initiated requests are buffered at the server side. Furthermore, the sensor switches to either listening or sleeping state immediately after it finishes sending the data, therefore, T_a is always equal to 0.

IV. ADAPTIVE SENSOR STATE TRANSITION

In the various sensor models discussed above, transitions among states besides being triggered by the sensor value diverging from its representation at the server, also occur due to timeouts. 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 timeouts. In this section, we derive how the sensor sets these timeouts in order to maximize energy savings.

In AS and ALS models, a sensor must set a sleep time T_s after which it transitions to the active state and sends an update to the server. Since the 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 $T_s + T_{sa} + T_{rx} + T_{tx}$ should not be greater than D. Therefore, T_s for the AL model is chosen to be

$$T_s^{AL} \leq D - (T_{sa} + T_{rx} + T_{tx}).$$

Similarly, T_s for the ALS model is chosen to be

$$T_s^{ALS} \leq D - (T_s + T_{sl} + T_{la} + T_{rx} + T_{tx}).$$

In AL, AS and ALS model, T_a needs to be determined. We 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 [9] for details. Waking up a sensor at 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.

Let us assume that $f(t)$ is the probability of receiving any type of requests at any time instant t. Let $\pi(t)$ be the probability of being silent for t time units, i.e., there are no requests before t until a request arrives at time t. Since any incoming request means the end of the silent period, $\pi(t) = f(t)$. 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 timeout update request), then

$$\pi(t) = f(t) = \frac{1}{T_a + T_s}.$$

In this case, the expected energy consumption for a single silent period will be

$$\begin{aligned} E &= \int_0^{T_a} \pi(t) PC_a t dt \\ &+ \int_{T_a}^{T_a+T_s} \pi(t)[PC_a T_a + PC_s(t - T_a) + E_{sa}]dt \\ &= \frac{PC_a T_a^2 + 2PC_a T_s T_a + (PC_s T_s^2 + 2E_{sa} T_s)}{2(T_a + T_s)} \end{aligned}$$

Since $T_a \geq 0$ and $E > 0$, E is non-decreasing and is minimal when $T_a = 0$. This means 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. Periodically, 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 i^{th} interval. The i^{th} 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: $t_0, \cdots, t_{n-1}$. We estimate the distribution π by the distribution which generates an idle period of length t_i with probability c_i/w for $\forall i \in \{0, \cdots, n-1\}$. $\sum_{i=0}^{n-1} c_i = w$. Thus T_a is chosen to be the value t_m that minimizes the energy consumption as follows:

$$\min_{t_m}\{\sum_{j=1}^{m-1} \frac{c_j}{w} PC_a t_j + \sum_{j=m}^{n} \frac{c_j}{w}[PC_a t_m + PC_s(t_j - t_m) + E_{sa}]\}$$

Similar derivation can be done to obtain T_a for AL/ALS models and T_l for the ALS model. (See details in [9]).

V. 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.

The Simulation Environment

We built a simulator in C, consisting of a server and a database and a number of sensors. User queries are posed at the server which then returns their results.

Symbol	value	symbol	value
PC_a	$14.88mW$	E_{sl}	$0.025\mu J$
PC_l	$12.50mW$	E_{la}	$0.014\mu J$
PC_s	$0.016mW$	E_{sa}	$0.119\mu J$
T_{sl}	$4\mu s$	T_{rx}	$0.05ms$
T_{la}	$12\mu s$	T_{tx}	$1.67ms$
T_{sa}	$16\mu s$		

TABLE VI
Parameters Used in the Simulation

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 [6]. 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 seconds. Each query is accompanied by an accuracy constraint specifying the maximum acceptable width of the result. The accuracy constraint 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 $A_{max} = A_{avg} \cdot (1 + A_{var})$.

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. 5 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 VI, listening state consumes similar amount of power to active state, thus the AL model does not decrease energy

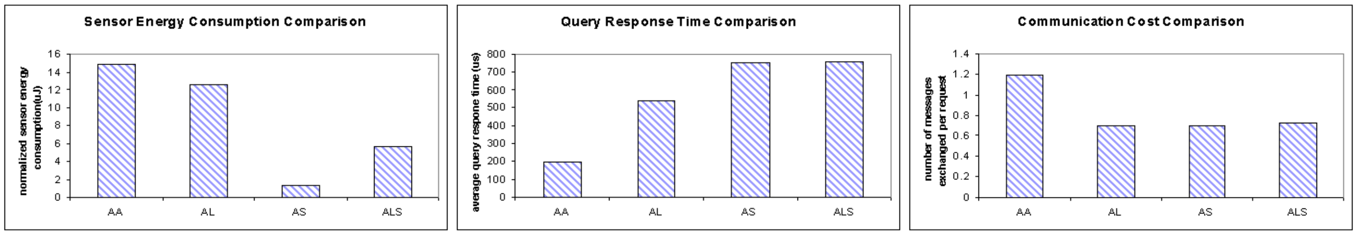


Fig. 5. System performance comparison of proposed sensor models

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. 6 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 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 timeout 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. 7 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 it value.

Fig. 8 depicts the impact of range size by showing 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. (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. 8 shows that our adaptive approach significantly outperforms other approaches.

Fig. 9 shows how the system behaves as sensor values change frequency varies. 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. 10 indicates the impact of query accuracy constraints. 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 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 timeout updates consume most of the energy. For this reason, the curve levels off after that point.

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

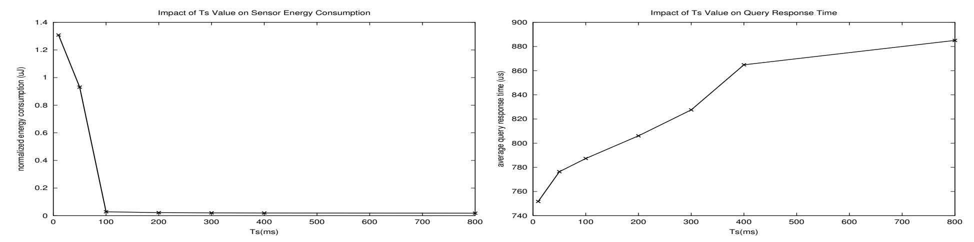


Fig. 6. Impact of T_s value on system performance

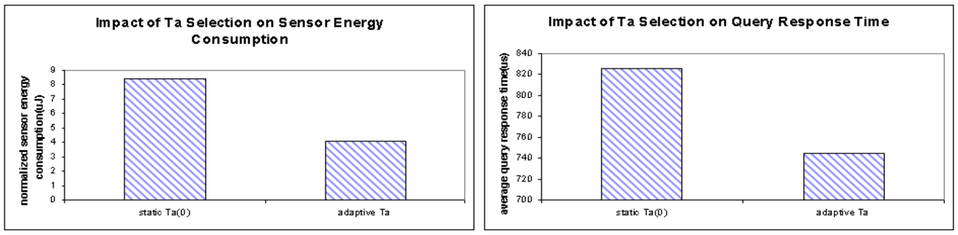


Fig. 7. Impact of T_a adaptation on system performance

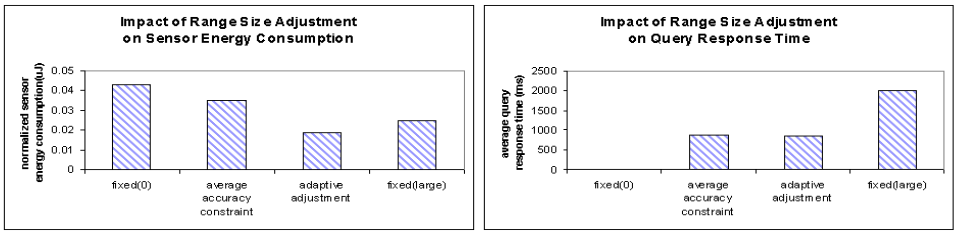


Fig. 8. Impact of range size adaptation on system performance

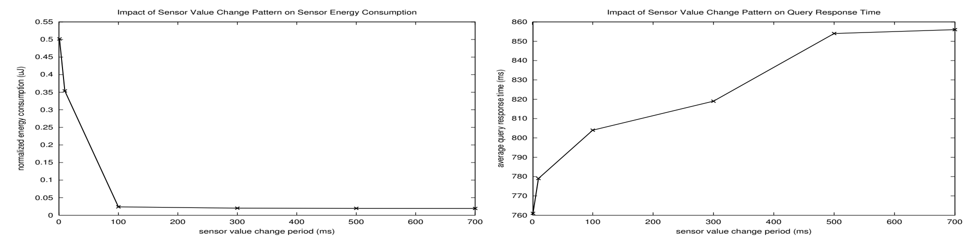


Fig. 9. Impact of sensor value changes on system performance

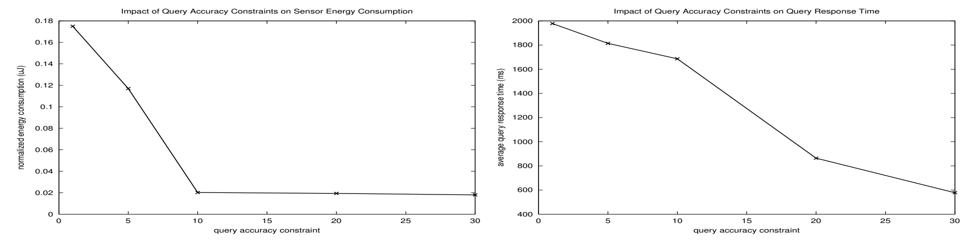


Fig. 10. Impact of query accuracy constraints on system performance

models and saves more energy than static range or storing exact values.

VI. Related Work

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 operation, e.g. transition less active into a sleeping mode. In ASCENT [10], 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 [11] protocol avoids the overhead of keeping neighbor state by letting each node probe its surrounding nodes to maintain a desired working node density. STEM [12] is a topology management scheme for sensor networks that trades power savings for path setup latency. Here a separate radio operates at a lower duty cycle to detect incoming packets. LEACH [13] and PEGASIS [14] aim to balance energy consumption among sensor nodes so that the lifetime of the sensor network can be maximized. The approaches described above focus on exploiting the cooperation among sensor nodes for better energy conservation. In contrast, our energy saving approaches takes into consideration application-level information to optimize sensor energy consumption.

Quality-aware data collection protocol is proposed in [2] with the objective of reducing communication overhead. For sensor environments, data archival strategies are studied to guarantee application quality in [4], where an optimal online algorithm is proposed for creating piecewise-constant approximation of sensor data. Offering precision and performance tradeoff for aggregate queries is done in [15]. When database values are not precise enough to answer aggregate queries, data producers need to be probed to improve the accuracy. Probing all of them would be very expensive, algorithms for selecting a subset of data producers to probe are investigated based on the cost of probing each data producer.

Research on sensor databases such as COUGAR [16] and Telegraph [17] 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 [18]. Similarly, Telegraph also studies adaptive query processing done in sensor network. In contrast to these approaches, our work explores how application quality tolerance can be used to reduce the energy consumption for query processing in sensor environments. The two approaches can be viewed as complementing each other – an interesting future work is to combine the in-network query processing with quality-based adaptation to minimize energy consumption. Other related work is research on data streaming [19], which investigates data management and processing problems in the presence of continuous and time varying data streams.

VII. CONCLUSIONS

Since many real-world applications can tolerate data imprecision at varying levels, the error tolerance of applications can be exploited to reduce energy consumption during sensor data collection. In this paper, we have studied energy efficient data collection mechanisms for distributed sensor environment that explores the tradeoff between sensor data accuracy and energy consumption. Both theoretical analysis and experimental results validated the effectiveness of our approaches.

To ensure the quality of aggregate queries, it is necessary to probe part but not all of the sensors. Our work can be further extended with a probabilistic cost model for probing sensors in order to efficiently handle aggregate queries. With the same goal of minimizing sensor energy consumption and ensuring application quality, dealing with aggregate continuous queries becomes much more complicated. Furthermore, in-network processing is a very promising approach to be integrated with our work.

REFERENCES

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramanian, and E. Cayirci, “Wireless sensor networks: A survey,” *Computer Networks*, vol. 38, no. 4, March 2002.

[2] C. Olston, B. T. Loo, and J. Widom, “Adaptive precision setting for cached approximate values,” in *ACM SIGMOD*, 2001.

[3] X. Yu, K. Niyogi, S. Mehrotra, and N. Venkatasubramanian, “Adaptive middleware for distributed sensor environments,” *IEEE DS Online*, 2003.

[4] I. Lazaridis and S. Mehrotra, “Capturing sensor generated time series with quality guarantees,” in *IEEE ICDE*, 2003.

[5] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, “Optimizing sensor networks in the energy-latency-density design space,” *IEEE Tran. Mobile Computing*, vol. 1, no. 1, January-March 2002.

[6] RT Monolithics Inc., http://www.rfm.com/, *ASH Transceiver TR1000 Data Sheet*.

[7] A. Sinha and A. Chandrakasan, “Dynamic power management in wireless sensor networks,” *IEEE Design and Test of Computers*, vol. 18, no. 2, March/April 2001.

[8] A. Chandrakasan, R. Min, M. Bhardwaj, S. Cho, and A. Wang, “Power aware wireless microsensor systems,” in *ESSCIRC*, 2002.

[9] Q. Han, Sharad Mehrotra, and N. Venkatasubramanian, “Energy efficient data collection in distributed sensor environments,” Tech. Rep., University of California-Irvine, 2003, http://www.ics.uci.edu/˜qhan/papers/sensor-techrep.pdf.

[10] A. Cerpa and D. Estrin, “Ascent: Adaptive self-configuring sensor networks topoligies,” in *InfoCom*, 2002.

[11] F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang, “Peas: A robust energy conserving protocol for long-lived sensor networks,” in *IEEE ICDCS*, 2003.

[12] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, “Topology management for sensor networks: Exploiting latency and density,” in *ACM MobiHoc*, 2002.

[13] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-efficient communication protocol for wireless microsensor networks,” in *HICSS*, 2000.

[14] S. Lindsey, C. Raghavendra, and K. M. Sivalingam, “Data gathering algorithms in sensor networks using energy metrics,” *IEEE Tran. Parallel and Distributed Systems*, vol. 13, no. 9, September 2002.

[15] C. Olston and J. Widom, “Offering a precision-performance tradeoff for aggregation queries over replicated data,” in *VLDB*, 2000.

[16] Philippe Bonnet, J. E. Gehrke, , and Praveen Seshadri, “Towards sensor database systems,” in *MDM*, 2001.

[17] S. R. Madden and M. J. Franklin, “Fjording the stream: An architecture for queries over streaming sensor data,” in *IEEE ICDE*, 2002.

[18] Y. Yao and J. Gehrke, “Query processing for sensor networks,” in *CIDR*, 2003.

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