Available online at www.sciencedirect.com

ScienceDirect

Information Sciences 178 (2008) 2128–2149

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Aggregation in sensor networks with a user-provided quality of service goal

Zhen He [a,1], Byung Suk Lee [b], X. Sean Wang [b,*]

[a] *Department of Computer Science, La Trobe University, Bundoora, VIC 3086, Australia*
[b] *Department of Computer Science, University of Vermont, Burlington, VT 05405, USA*

Received 25 August 2006; received in revised form 6 January 2008; accepted 7 January 2008

Abstract

Significant research has been devoted to aggregation in sensor networks with the purpose of optimizing its performance. Existing research has mostly concentrated on maximizing network lifetime within a user-given error bound. In general, the greater the error bound, the longer the lifetime. However, in some situations, it may not be realistic for the user to provide an error bound. Instead, the user may want to provide a quality of service (QoS) goal that has a combined objective of lifetime and error. Indeed, the error tolerable by the user may depend on how long the sensor network can last. This paper presents an aggregation protocol and related algorithms for reaching such a QoS goal. The key idea is to periodically modify a filter threshold for each sensor in a way that is optimal within the user objective, the key technical method being to translate the problem into a mathematical programming formulation with constraints coming from various sources, such as the user, the sensor network, and the data characteristics. Extensive experiments demonstrate the high accuracy, high flexibility and low overhead of this QoS-based optimization approach.
© 2008 Elsevier Inc. All rights reserved.

Keywords: Sensor networks; Aggregation; Quality of service; Distributed optimization

1. Introduction

Recently, there has been significant research conducted on data aggregation in wireless sensor networks. Two performance factors have been considered: sensor network lifetime and aggregation precision (or error). Although the approaches vary, existing work is mostly concerned with maximizing lifetime under certain error tolerance. (The existing work will be discussed in Section 2.) Some users, however, may have a more flexible objective. For instance, they may want the opposite, that is, to minimize error under certain lifetime constraints. This may be necessary in a situation where precision of data is important and there is limited useful lifetime. Furthermore, users may want to achieve a more complex objective, such as to maximize lifetime and minimize error under

* Corresponding author. Tel.: +1 802 656 3342; fax: +1 802 656 0696.
E-mail addresses: z.he@latrobe.edu.au (Z. He), Byung.Lee@uvm.ed (B.S. Lee), Sean.Wang@uvm.edu (X.S. Wang).
1 Part of this author's work was done while he was at the Department of Computer Science, University of Vermont.

0020-0255/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2008.01.003

certain constraints on both error and lifetime. This may be necessary in a situation where users' error tolerance depends on the possible lifetime. We refer to the above performance objectives as a quality of service (QoS) goal.

In the framework of this paper, we allow users to express their QoS goal by providing an objective function with lifetime and error as the variables, together with constraints in the form of lifetime and error bounds. The performance goal of the sensor network, then, is to maximize the QoS achieved within the user-provided constraints. (If the error and lifetime bounds are found to be too restrictive to be feasible, the system may go back to the user for more relaxed constraints.)

In order to maximize the QoS, the trade-off between lifetime and error needs to be controlled. The basic mechanism we use is to adjust a *filter threshold* for each sensor and have each sensor operate as follows: if the value read (gathered, collected, or sensed) by a sensor is within the filter threshold given to the sensor, the sensor does not send data, thus saving battery power needed for transmission. (This is the same as the "filter approach" described in Olson et al. [19].) The aggregation error is aggregated from the filter thresholds of the non-sending sensors.

With the above mechanism, in order to reach the performance goal, the filter thresholds need to be adjusted toward maximizing the QoS. Here, we assume that the lifetime of a sensor is solely dependent on how much data it has to transmit. The more data it has to send, the shorter the life, and vice versa. Therefore, the larger the filter thresholds, the longer the lifetimes of the sensors, but the higher the aggregation error.

Our approach is to develop a mathematical formulation of an optimization problem whose solution gives the best filter thresholds to sensors. In this formulation, in addition to the above *user-provided* constraints, there are also *system-induced* constraints from the sensor network system, due to its configuration and operation strategy (including a cluster head rotation policy). Furthermore, the amount of transmission energy which can be saved (hence lifetime gained) by a particular filter threshold also depends on the characteristics of the data. In this paper, we use historical data to predict the impact on future energy savings by specific filter thresholds. We refer to the corresponding constraints as the *data-dependent* constraints.

Once we know the user-provided, system-induced and data-dependent constraints, we have an optimization problem in which the optimization parameters are the filter thresholds for the sensors. Since the data characteristics and network status (e.g., actual battery consumption) may change over time, we need to perform the above optimization periodically, to reflect precisely the current status of the sensor network. For the purpose of such periodic adjustments, we use a general protocol that allows the system to collect the parameters for all the relevant constraints and to disseminate the optimized filter thresholds.

We demonstrate our method through a detailed algorithm design, based on the above framework, by assuming a commonly-used sensor network configuration. In the implementation of algorithms, we consider SUM as the aggregation function. We also show results from extensive experiments with real data sets. The experiment results show that the optimization computation using our method is very accurate, despite the approximations made in our implementations and can flexibly adjust to different QoS goals (expressed as objective functions). Additionally, we compare our algorithm with the most comparable existing method in which the user-provided objective is simply to maximize lifetime (subject to an error bound). The results show that our method produces almost the same optimal solutions (e.g., similar lifetime for the same error bound) despite the overhead inherent in making our method more flexible and versatile in handling general QoS goals.

The main contributions of this paper include: (1) the introduction of the concept of flexible user-provided QoS goals; (2) the optimization formulation of the problem in meeting the QoS goals; (3) the aggregation query processing protocol supporting the optimization formulation; and (4) an instantiation of the protocol under realistic assumptions.

The remainder of the paper is organized as follows: Section 2 discusses related work; Section 3 introduces an aggregation protocol for our optimization problem; Section 3.3 presents generic algorithms based on the protocol; Section 4 describes a specific instantiation of the generic algorithms; Section 5 evaluates the instantiated algorithms and Section 6 concludes the paper.

2. Related work

The existing work on sensor network aggregations aims to maximize lifetime. We classify the existing methods into three categories, based on the error tolerance type: zero-error [13,14,23,24], user-defined-error

[7,19,21], and low-error [6,12,16,18,22]. It should be noted that some research has focused on ensuring certain QoS constraints (where QoS is defined in terms of the latency of delivering aggregates from source to sink nodes) are met [27,29,28]. However, our work is more focused on the quality of the collected data, rather than the delivery speed of the data.

In the zero-error category, Madden et al. [14] and Yao et al. [24] provide methods to maximize lifetime by performing in-network aggregation while using a tree-based routing model. Kalpakis et al. [13] study a problem in which, given a collection of sensors and a sink, as well as their locations and the energy of each sensor. They find a data collection schedule that maximizes lifetime. Tan et al. [23] develop a routing scheme that maximizes lifetime for a given set of sensors. Fan et al. [8] develop a structure-free aggregation algorithm that maximizes lifetime for a given set of sensors. These studies differ from ours in that they do not allow any imprecision in the aggregation and, therefore, do not allow users to trade accuracy for extended lifetime.

In the user-defined-error category, Olston et al. [19] propose an adaptive filter-based approach for a single-hop network. The filters adapt to changing conditions to minimize the data sent, while ensuring that the user-defined error bound is not violated. Deligiannakis et al. [7] adapt the method by Olston et al. [19] to work with a tree-based routing model. Sharaf et al. [21] maximize lifetime by influencing the routing tree construction to reduce the transmitted data and imposing a hierarchy of output filters on the sensor network. This aforementioned work allows users to trade accuracy for extended lifetime (within the error bound) but, unlike our work, does not allow users to specify a *combined* lifetime and error goal. Work by Ren and Liang [20] probabilistically (not deterministically) guarantees the user-defined error bound, given a confidence interval on the accuracy, while reducing the energy consumption to lengthen the lifetime.

In the low-error category, Considine et al. [6] and Nath et al. [18] provide duplicate-insensitive sketches and synopses, respectively, to perform approximate in-network aggregation. Their methods are designed to work with a ring-based, multi-path routing model, which means errors caused by communication failures are greatly minimized. Amit et al. [16] combine the advantages of tree routing and multi-path routing by running them simultaneously in different regions of the network. Their method reduced error caused by packet loss by up to three times more than all previous methods. Sharifzadeh et al. [22] aggregate sensor readings while taking spatial distribution of the sensor nodes into consideration. Kapalpkis et al. [12] maximize lifetime for aggregate range queries using linear sketches. However, this previously mentioned work, unlike ours does not allow users to control how accuracy should be traded for extended lifetime. Instead, they use various heuristics and probabilistic mechanisms to keep the error low, while maximizing lifetime.

In contrast with the research above, Madden et al. [15] allow users to specify the minimum *lifetime* constraint. The sampling rate is then estimated so the lifetime constraint is satisfied. The sampling rate directly influences the data precision. This work differs from ours in two respects: firstly, our work adjusts the transmission of sampled data, not the sampling itself; and secondly, we allow the user to set a more flexible QoS goal.

3. Aggregation protocols

In this section, we describe the aggregation protocols in a generic way, that is, the presented protocols are not specific to any particular routing strategy or aggregation hierarchy. We also provide generic algorithms for key steps of the protocol. (In Section 4, we will present specific algorithms designed for a cluster-based routing strategy and a corresponding aggregation hierarchy.)

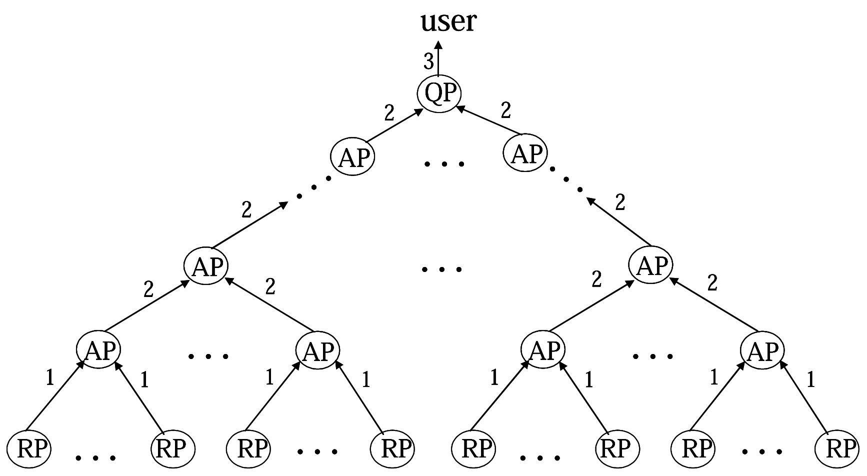
3.1. An overview

Underlying the optimization performed in our aggregation protocol is the use of *filter thresholds*. A filter threshold is defined as the range between the lower bound and upper bound on sensor readings so that a sensor sends a reading only if it falls outside the range. The optimization is to periodically adjust the filter thresholds of individual sensors, in order to achieve certain user-requested QoS goals under system-induced and data-dependent constraints globally at the network level. By adjusting the filter thresholds, we can adjust the balance between the aggregation error and the network lifetime.

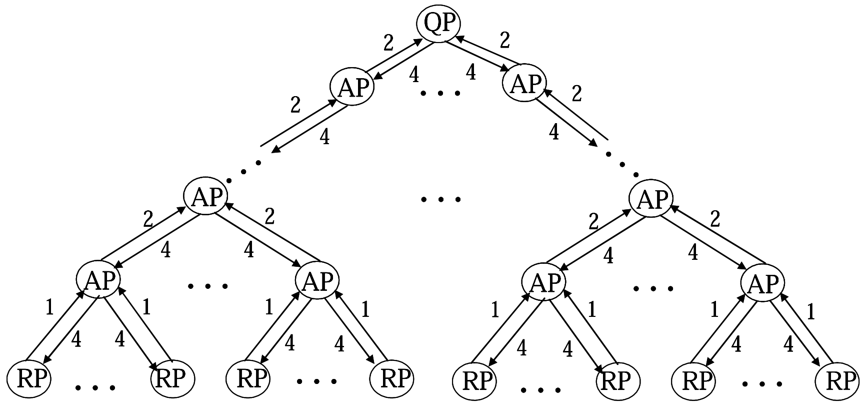
A sensor assumes one or more of the following roles for aggregation query processing: a reading point (RP), an aggregation point (AP), and a query point (QP). Sensors in these roles configure an aggregation hierarchy as shown in Fig. 1. Note that this aggregation hierarchy is generic in the sense that it is independent of the routing model. That is, the same hierarchy can be placed on top of any routing model, such as tree-based routing, cluster-based routing, etc. For instance, an AP may be a non-leaf node of the tree in the tree-based routing or a cluster head in the cluster-based routing. There can be multiple QPs, thus multiple aggregation hierarchies, in the same network.

The protocol has two operation modes: a *normal mode* and an *update mode*. In the normal mode, aggregations of readings are done. The protocol is summarized below (see Fig. 1a). We assume the readings from different sensors are synchronized. (Synchronization in itself is a research issue and is beyond the scope of this paper.)

Step 1. An RP sends readings, if outside the filter threshold, to its AP. Each time a reading is sent, the filter threshold is centered on the reading. This centering is done to increase the probability that the next reading will be within the filter threshold, assuming that sensor readings do not change abruptly (see Example 1 below).



(a) Normal mode operation.



(b) Update mode operation.

Fig. 1. Aggregation hierarchy and protocol.

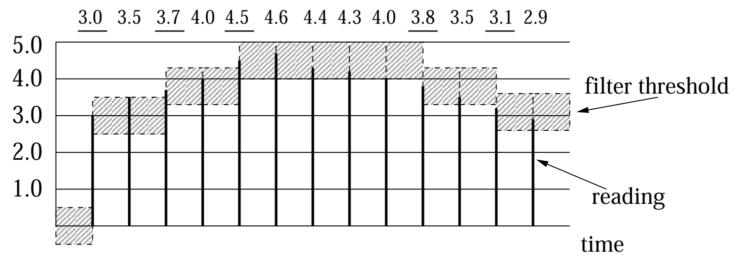
Step 2. The AP generates a partial aggregation of the readings and sends it to the next level in the hierarchy. For a reading not sent by an RP, a first-level AP assumes the reading is the same as the last reading; for a reading sent by an RP, it uses the reading sent.

Step 3. The QP generates a total aggregation of the readings and reports it to the user.

3.2. Update mode protocol

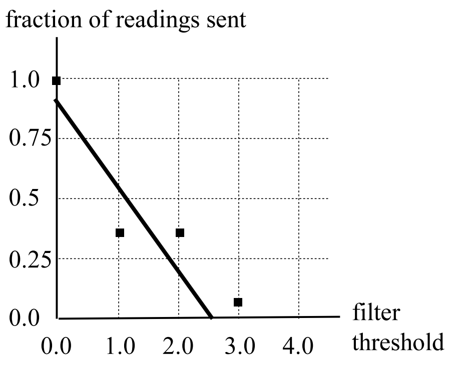
One of the key challenges in the update mode is to devise a way of capturing the characteristics of the data being collected at each RP. This information allows the RP to determine the number of readings that need to be sent to the QP for a given filter threshold. To accomplish this, we use what we call a *filter Threshold–Reading fraction (T–R) curve*. A *T–R* curve is generated by each RP and characterizes the number of readings to be sent, given a particular setting of the filter threshold.

Definition 1 (*T–R curve*). Consider a set of pairs $\langle \epsilon_j, r_j \rangle$ in the space $T \times R$, where $\epsilon_j \in T$ ($\epsilon_j \geqslant 0$) is a filter threshold and $r_j \in R$ ($r_j > 0$) is the fraction of the readings that, given the value of T, would be sent out among



This example shows that when the filter threshold is 1.0, the RP will need to send the readings 3.0, 3.7, 4.5, 3.8, and 3.1, while re-centering the filter threshold after sending each reading. The readings to be sent are underlined in the figure.

(a) Generating a set of T-R data points.



This example shows generating a T-R curve through linear regression of the data points generated in (a). In this example the T-R curve is a straight line fitting the five data points with the least sum of squares errors (see Section 4.2).

(b) Fitting a T-R curve to T-R data points.

Fig. 2. An example of generating a *T–R* curve.

all the readings made by an RP during the previous update interval. Then, a *T–R* curve is defined as a function in a family of functions from T to R such that $\sum_j (r_j - f(\epsilon_j))^2$ is minimized.

The example below shows how a *T–R* curve is generated.

Example 1. Suppose the 13 readings shown at the top of Fig. 2a were made by an RP. Then, the RP counts the number of readings that would be sent for each of the filter thresholds from 0.0 with an increment of 1.0, and finds that 13 points, five points, three points, three points, and one point would be sent if the filter thresholds were 0.0, 1.0, 2.0, 3.0, and 4.0, respectively. The iteration stops when the number of readings to be sent becomes 1 (which is the smallest number that renders the reading fraction (r_j in Definition 1) greater than zero). As a result, the RP collects a set of five data points, {(0.0, 13/13), (1.0, 5/13), (2.0, 3/13), (3.0, 3/13), (4.0, 1/13)}. Then, a *T–R* curve may be generated from these data points through regression. Fig. 2b shows the case of using *linear* regression.

In the update mode, optimal filter thresholds are computed by QP and used as the new filter thresholds of each RP. This is done at a predefined update interval. The protocol can be summarized as follows: (see Fig. 1b).

Step 1. An RP generates a *T–R* curve (using the stored sample readings acquired during the update interval) and sends it to its AP.
Step 2. The AP forwards the *T–R* curves to the next level in the hierarchy.
Step 3. The QP computes optimal filter thresholds for each RP based on all the *T–R* curves received, the objective function, and the user-provided and system-induced constraints.
Step 4. The QP sends the filter thresholds to the RPs (through APs).

3.3. Generic protocol algorithms

The algorithms for the normal mode steps are straightforward. Thus, here we provide the algorithms for the update mode protocol only, specifically for generating *T–R* curves and computing optimal filter thresholds.

```
Algorithm Generate_TR_curve
// Executed by each RP.
Inputs:
    • $U$: update interval.
    • $\rho$: sampling rate of the RP.
    • $S$: a sequence of readings measured by the RP during the previous $U$.
Output:
    • mapping $f : T \to R$, where $R$ is the fraction of the readings that need to be sent per
      sample interval for a given filter threshold $T$.
Procedure:
begin
    1. Generate a set of pairs $\langle \epsilon_j, r_j \rangle$ where $\epsilon_j$ is a filter threshold and $r_j$ is a fraction of the
       readings in $S$ that would have been sent, given $\epsilon_j$. Here, the fraction is computed relative
       to $length(U) * \rho$, i.e., the number of readings during $U$.
    2. Find a mapping from $T$ to $R$ as a function $R = f(T)$, such that $\sum_j (r_j - f(\epsilon_j))^2$ is the
       minimum, where $\langle \epsilon_j, r_j \rangle$ is the $j^{th}$ pair collected in Step 1.
end
```

Fig. 3. Algorithm Generate_TR_curve.

```
Algorithm Compute_T
// Executed by QP.
Inputs:
    • objective function $h(L, E)$, where $L$ is network lifetime and $E$ is aggregation error.
    • user-provided constraints.
    • system parameters.
    • T-R curves from all RPs (generated during the last update interval).
Output:
    • new filter threshold $T_i$ for each reading point RP$_i$.
Procedure:
begin
    1. For each RP$_i$ ($i = 1, 2, \ldots, m$), update its running average T-R curve with the input T-R
       curve, and use the updated curve to produce T-R constraints.
    2. Generate system-induced constraints using the input system parameters.
    3. Find the pair of $Lopt$ and $Eopt$ that maximizes $h(L, E)$ while satisfying the user-provided
       constraints (input), the system-induced constraints (Step 2), and the T-R constraints
       (Step 1). Then, for each RP$_i$ ($i = 1, 2, \ldots, m$), return the $T_i$ that corresponds to $Lopt$
       and $Eopt$.
end
```

Fig. 4. Algorithm Compute_T.

Fig. 3 shows the generic algorithm Generate_TR_curve for each RP to generate a *T–R* curve. The basic idea is for each RP to store the sampled readings for the current update interval. Then, at the filter threshold update time, the sampled readings are used to generate a *T–R* curve. This approach reduces the amount of data sent from the RPs to the QP by sending only a curve that describes the relevant characteristics of the data, instead of the individual readings. In this algorithm, we assume that the sampling rate (ρ) is constant during an update interval (U). The output is a *T–R* curve, defined in Definition 1.

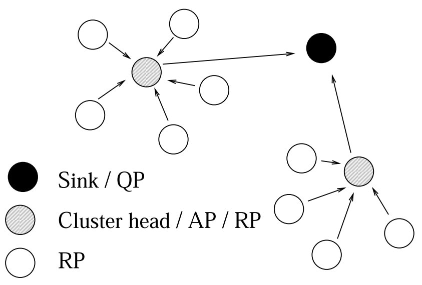
Fig. 4 shows the generic algorithm Compute_T. The basic idea is to produce a set of constraints comprising of data-dependent (i.e., *T–R*) constraints (in Step 1) and system-induced constraints (in Step 2), combined with user-provided objective function and any additional user-provided constraints. Using these functions and constraints, QP computes the optimal lifetime and aggregation error as well as the optimal filter thresholds to be sent to the individual RPs.

In Step 1, the new *T–R* curve generated by an RP for the last update interval is used to update the running average of the *T–R* curves for the RP. This is done separately for each RP. Using a running average instead of only the new *T–R* curve will allow the long term pattern of readings to be reflected. In Step 2, the system parameters include the costs of sampling, sending and receiving a reading, the remaining battery power, etc. Some of these parameters are static (e.g., costs of sampling) and some are dynamic (e.g., battery power). In Step 3, if the user-provided constraints turn out to be infeasible (i.e., no solution possible), then the user should be consulted to relax the constraints.

4. Algorithm instantiation

4.1. Setup for the instantiation

In our instantiation, we consider *cluster-based* routing, illustrated in Fig. 5, in a multi-hop network configuration. This routing approach has been widely studied in the networking community [1–4,9,10,17,25,26]. In



This figure shows two clusters, each with one cluster head.

Fig. 5. Cluster-based routing.

this approach, nodes with geographical proximity form a cluster, after which one node is elected to be the cluster head in each cluster. Each node (other than the sink) will be an RP, while the cluster head assumes the dual roles of AP and RP. A cluster head forwards packets sent by RPs toward the destination sink node, namely QP. In order to balance the load for data transmission among the sensors, a new cluster head is selected periodically. We assume that each sensor is selected as a cluster head with an *equal frequency* which results in all sensors consuming the same battery power.

Additionally, we assume sensor readings (i.e., samples) are *synchronized*, as mentioned in Section 3, and the lifetime of the sensor network is the time until the *first* sensor runs out of battery power.

We use *linear programming* as the optimization technique for its computational efficiency. For this purpose, we use a linear function to approximate a *T–R* constraint using a *T–R* curve. Additionally, we use a linear function to compute the aggregation error. This is done by computing it as the sum of *all* the filter thresholds, that is, as the aggregation error (upper) bound.[2] Determining the filter thresholds of only the non-sending sensors requires taking a *T–R* curve into consideration for computing the aggregation error and, as a result, makes the function non-linear.

Furthermore, we require the objective function $h(L,E)$ (of the network lifetime L and the aggregation error E) to be *monotonic*. That is, $h(L,E)$ should be monotonically non-increasing when E increases with a fixed L, and monotonically non-decreasing when L increases with a fixed E. This property fits the premise that a user's QoS goal is better achieved if the error is lower and/or the lifetime is longer. For instance, any function of the form $cf(L) - g(E)$ or $cf(L)/g(E)$, where $f(L)$ is monotonically non-decreasing with L and $g(E)$ is monotonically non-increasing with E (and c is a calibration coefficient), will flexibly do as $h(L,E)$. All of the following example functions have this monotonicity property and are suitable as $h(L,E)$: $cL - E$, $c\log L - E$, cL/E, L, $-E$, $1/E$. The optimization step (Step 3 of Compute_T) will take advantage of this monotonicity property.

4.2. Instantiation of Generate_TR_curve

A key design decision in the instantiation of the generic algorithm is the use of linear regression to produce the *T–R* curve. We use linear regression for the following reasons: (1) it is computationally efficient; (2) the resulting equation requires very small storage space (only two regression coefficients), thus can be sent up the aggregation hierarchy at little cost; and (3) it facilitates the linear programming approach used for optimization in Compute_T.

Fig. 6 shows an instantiation of Step 1 of the algorithm Generate_TR_curve. In the algorithm, we use an additional input d_T, the increment of a filter threshold, so that Step 3 of Fig. 6 iterates for ϵ_i from 0.0 at the increment of d_T. The iteration stops once the number of readings to be sent (c_j) drops to equal to or less than 1.

[2] The aggregation error computed in [19], which is compared against our method in Section 5, is also an upper bound.

```
1. Compute the total number ($N$) of readings during $U$ as $length(U) * \rho$.
2. Initialize $Dset$ to an empty set.
3. For each $\epsilon_j$ from 0.0 at the increment of $d_T$ {
    (a) Count the number ($c_j$) of readings that would be sent given $\epsilon_j$.
    (b) Compute the ratio $r_j$ as $r_j = c_j/N$.
    (c) Insert $\langle \epsilon_j, r_j \rangle$ into $Dset$.
   } until $c_j \leq 1$.
4. Return $Dset$.
```

Fig. 6. *T–R* data set generation step (Step 1) of Generate_TR_curve (instantiation).

4.3. Instantiation of Compute_T

In Step 1 of the algorithm Compute_T (see Fig. 4), the *T–R* curve generated by RP$_i$ ($i = 1, 2, \ldots, m$) is $R_i = f_i(T_i) = a_i + b_i \times T_i$. To generate a running average of the *T–R* curves for each RP$_i$, QP only needs to obtain the average values $\bar{a}_i$ and $\bar{b}_i$ from all the a_i and b_i values sent by RP$_i$ in the past. This can be done incrementally by QP with negligible overhead.

Using the running average *T–R* curve, we generate a *T–R constraint* for RP$_i$ as follows:

$$T_i = (R_i - \bar{a}_i)/\bar{b}_i \tag{1}$$

where T_i and R_i are the filter threshold and reading fraction of RP$_i$, respectively. Note that the constant $\bar{b}_i$ is a *negative* number for all reading points RP$_i$ ($i = 1, 2, \ldots, m$). This is intuitive because the filter threshold T_i should decrease as R_i increases, i.e., as more readings are sent by RP$_i$.

As mentioned earlier, we use the *aggregation error bound*, *E*, for our computation of an aggregation error:

$$E = \sum_{i=1}^{m} T_i \tag{2}$$

where $T_1, T_2, \ldots, T_m$ are all the RPs. Accordingly, in Step 3 (the optimization step described below) we aim to minimize the aggregation error bound E by adjusting the filter thresholds while satisfying certain system-induced, data-dependent and user-provided constraints. As mentioned earlier, we accomplish this by using linear programming.

In Step 2 of the algorithm Compute_T, given a sensor network with m reading points (RP$_i$, $i = 1, 2, \ldots, m$), the system-induced constraints are:

$$R_i = g(L, i) - \sum_{k \in C(i)} (w_{ik} R_{ik}) \quad \text{for } i = 1, 2, \ldots, m \tag{3}$$

where L is the lifetime of the sensor network, $g(L, i)$ is a system-dependent function of L, and w_{ik} and R_{ik} are, respectively, the weight and the fraction of readings sent to RP$_i$ by RP$_k$ that is in the same cluster as RP$_i$, denoted as $C(i)$. Here, $C(i)$ does not include RP$_i$ because, as shown in the Appendix, the weight w_{ik} is derived from the battery power consumed by RP$_i$ while it is a cluster head, thus receiving messages from its member nodes RP$_k$ ($k \neq i$). The specific g and w_{ik} depend on factors like the cost of sending and receiving a reading, the routing model, etc. We show the details of deriving Eq. (3) in the Appendix. Fig. 7 shows our instantiation of Step 3 of the algorithm Compute_T. We use two additional inputs in our instantiation. One input is d_L, the increment of a lifetime L at each iteration; the other is $L_{\max}$, the maximum value of L used in the iteration. In the algorithm, we compute the optimization for *each value* of L during the iteration. The reason for fixing the value of L is that $g(L, i)$ of Eq. (3) is not a linear function of L; it becomes linear (with R_{ik}'s) once L is fixed, which in turn makes Eq. (3) applicable to linear programming as a linear constraint.

```
1. Repeat from $L = d_L$ at the increment of $d_L$ until $L > Lmax$ {
    (a) Use the simplex algorithm to find the minimum $E$, $Emin$, given user-provided constraints, T-R constraints (Equation 1), and system-induced constraints (Equation 3).
    (b) Evaluate $h(L, Emin)$.
   } while keeping track of the pair of $L$ and $Emin$ that gives the highest value of $h(L, Emin)$.
2. Set $\langle Lopt, Eopt\rangle$ to $\langle L, Emin\rangle$ resulting from the previous step, and return the values of $E_1, E_2, \cdots, E_m$ corresponding to $Eopt$.
```

Fig. 7. Optimization step (Step 3) of Compute_T (instantiation).

The initial value of L in the iteration is d_L, which we regard as the minimum possible lifetime with respect to the increment unit d_L. (The value of L cannot be zero in our optimization, as $g(L, i)$ is infinity when L equals zero (see Eq. (9)).) The iteration then runs at the increment of d_L and stops once L becomes greater than $L_{\max}$. At each iteration, the minimum E ($E_{\min}$) is found for the current value of L, as the objective is to minimize E for the given L due to the monotonicity property required of $h(L, E)$. We use the classic linear programming algorithm, simplex, for this. The resulting pair of L and $E_{\min}$ at the end of the iterations gives the optimal pair $\langle L_{\text{opt}}, E_{\text{opt}}\rangle$.

The procedure for finding L_{opt} and E_{opt} also gives the filter thresholds for the individual RPs. Indeed, when solving the linear programming problem for each value of L, the value E_i for each RP$_i$ is computed as well. The set of E_i computed for L_{opt} is the set of filter thresholds assigned to individual RPs.

5. Performance evaluations

We conduct three sets of experiments with the objectives of evaluating the *accuracy*, *flexibility*, and *overhead* of the optimization computations of our method. The accuracy experiments aim to determine the impact of approximations made in order to use linear programming as the optimization technique in the implementations (see Eqs. (1)–(3)). The flexibility experiments aim to determine how flexibly our method behaves according to the user's QoS goal. The overhead experiments aim to determine how our method compares to existing methods, with the overhead of sending *T–R* curves and newly optimized filter thresholds between RPs and QP.

5.1. Simulation setup

We conduct our experiments on the sensor network simulator and emulator (SENSE) [5]. The simulated network has four layers: application, networking, MAC, and physical. We built our algorithms into the application layer and also performed the routing inside the application layer rather than the network layer, which is the traditional approach as the behaviors of our algorithms are highly dependent on routing decisions.

The routing algorithm simulated is the simple cluster-based routing (see Fig. 5). For this, we first create the cluster heads randomly and then assign randomly created sensors to the closest cluster head. All sensors transmit to the sink via one cluster head. The simulated network has 50 sensors, 10 clusters, and one sink in all experiments.

We use a simple MAC layer which places the packets onto the physical layer, and assume there is no packet loss. We, therefore, do not send acknowledgment packets. At the physical layer, we implement the same energy consumption model (for sending and receiving packets) as that used in the experiments described in [10].

Table 1 summarizes the simulation parameter settings used. E_{elec} is the energy consumed by the transmitter electronics per bit sent or received, and ε_{amp} is the energy consumed by the transmitter amplifier. The values used for E_{elec} and ε_{amp} are the same as those used in [26].

Table 1
Simulation parameter settings

Parameter	Setting
E_{elec}	5 nJ/bit
ε_{amp}	100 pJ/bit/m^2
Sensor placement area	500 × 500 m
Sampling interval	10 s
Cluster rotation interval	200 s
Initial energy of a sensor	0.5 J

Given these settings, energy used to send a packet from one node to another, E_{TX}, is computed as:

$$E_{\text{TX}}(b,d) = E_{\text{elec}} \times b + \varepsilon_{\text{amp}} \times b \times d^2 \tag{4}$$

where b is the number of bits of the message, and d is the distance between the source and destination nodes. Energy used by a node to receive a message, E_{RX}, is computed as:

$$E_{\text{RX}}(b) = E_{\text{elec}} \times b \tag{5}$$

5.2. Algorithm setup

Parameters used in our algorithms are set as follows: the update interval 200 s, d_T (in Fig. 6) 0.01, and d_L (in Fig. 7) 50.0. Other parameters whose settings vary depending on the experiment; will be mentioned when the experiments are presented. In the Filter method, the filter threshold is increased periodically. For a fair comparison, we have set the interval between successive increases to 200 s, which is the same as the update interval used in our QoS method.

5.3. Data sets

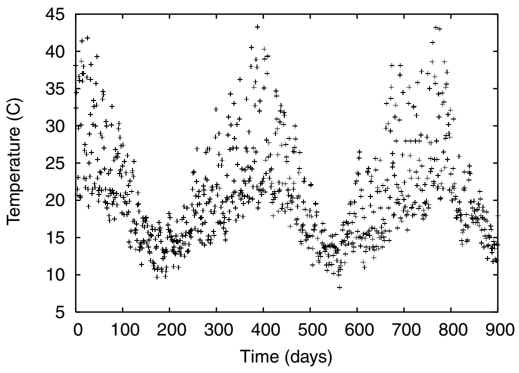
In our experiments, we use three time series data sets, shown in Fig. 8. We downscale the time in the data sets so that one day is mapped to 10 seconds. Then, for each original time-series data set, we generate 49 additional data sets by adding small variations to the data values using two different methods: *fixed offset* method and *random offset* method. A fixed offset method simulates a situation in which a sensor at one location always reads a value as either slightly larger or slightly smaller than a sensor at another location, as might happen if, for example, one sensor is a bit closer to a heat source. This is implemented by adding a constant increment to the previous reading. That is, given the original readings $v_1, v_2, \ldots, v_n$, the ith ($i = 1, 2, \ldots, 49$) data set has readings of $v_1 + 0.1 \times i, v_2 + 0.1 \times i, \ldots, v_n + 0.1 \times i$.

A random offset method simulates a situation in which the sensor location determines the variance of readings, for example, as might happen if one temperature sensor is in the shadow of a tree, while another is fully exposed, with the temperature reading affected by volatile wind conditions. The ith data set is generated as $v_1 + \text{rand}[0, 0.1 \times i], v_2 + \text{rand}[0, 0.1 \times i], \ldots, v_n + \text{rand}[0, 0.1 \times i]$ where $\text{rand}[0, 0.1 \times i]$ generates a random number between 0.0 and $0.1 \times i$ each time it is called. The resulting 50 time-series are used as the readings from 50 different sensors. Due to space limitations, only the experimental results for random offset data sets are presented. All results obtained using fixed offset data sets show similar trends.

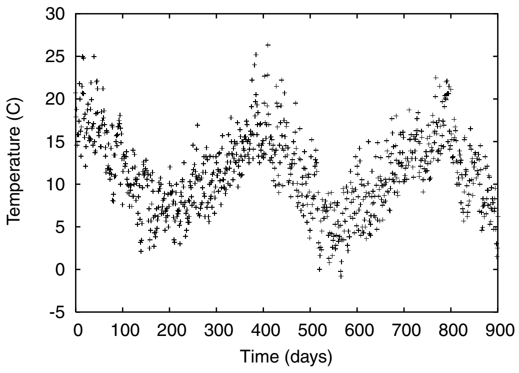
5.4. Experiments and the results

5.4.1. Accuracy of the optimization computations

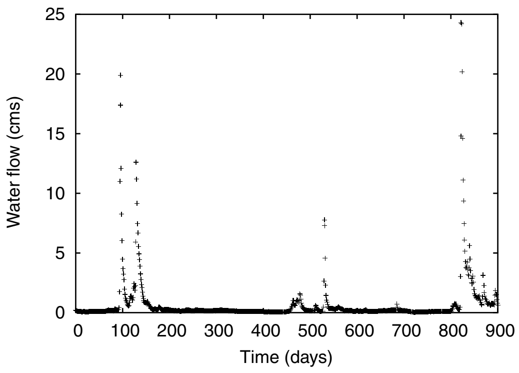
To evaluate the accuracy, we compare the projected lifetime and actual lifetime at each iteration of the optimization computation. The projected lifetime is recomputed (by the QP) at each iteration. We measure a lifetime as the time until the *first node* in the network dies. We set up our method to maximize the lifetime and minimize the aggregation error under both an error bound and a lifetime bound. Specifically, we use the



(a) Melb-max



(b) Melb-min



(c) Fisher[11]

Fig. 8. The original time-series data sets used in the experiments. (See above-mentioned references for further information.)

objective function $h(L,E) = 0.001L - E$, the error (upper) bound ($E_{\max}$) of 10.0 and the lifetime (lower) bound ($L_{\min}$) of 11,200 seconds.

Fig. 9 shows the results of this experiment. The results show that overall the projected lifetime and the actual lifetime are overall close, differing by less than 5% at the beginning and then converging towards the end of the iterations. This indicates that our optimization computations are quite accurate and, particularly, that our use of *linear T–R* curves is adequate for these data sets.

We have also tried varying parameters such as the update interval, the area of sensor field and the number of clusters. The results show that the accuracy does not vary much with these parameters. In all the cases we

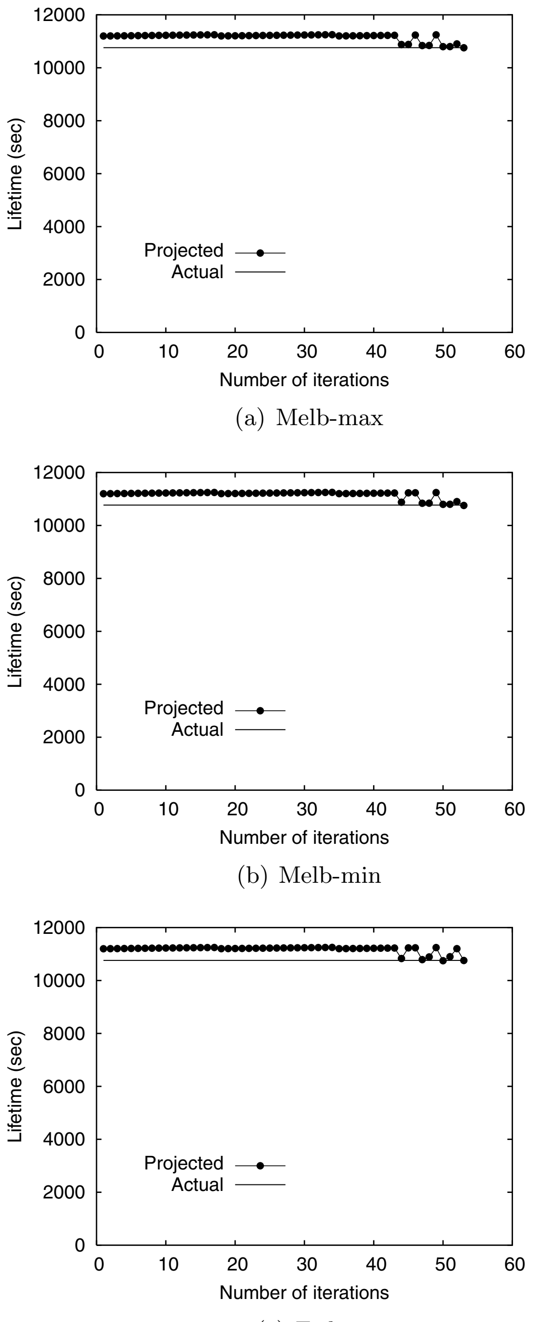


Fig. 9. Accuracy of optimization computation.

examined, the difference between the projected lifetime and the achieved lifetime is no more than 10%. All the performance graphs show two flat lines, as in Fig. 9. Thus, we have omitted these graphs.

5.4.2. *Flexibility of the optimization computations*

We set up our method with three different objective functions and the same set of user-provided constraints. Specifically, we use the following three cases of objective functions: $h(L,E) = -E$, $h(L,E) = L$, and

$h(L,E) = 0.01L - E$. Additionally, we use the following user-provided constraints: error (upper) bound ($E_{\max}$) of 10.0 and the lifetime (lower) bound ($L_{\min}$) of 10,500 s.[3] The achieved network lifetime is the time until the first node dies. The achieved aggregation error bound is computed as a temporal average of the sum of the filter thresholds of all sensors.

Fig. 10 shows the resulting pairs of the achieved lifetime and the achieved error for each objective function. With $h(L,E) = -E$, the achieved lifetime is compromised as much as possible toward the minimum allowed lifetime ($L_{\min}$) in order to minimize the achieved error. (The achieved lifetime is lower than $L_{\min}$ by about 2.4%; this is due to slight inaccuracy in the optimization computations (see Fig. 9).) The converse is true with $h(L,E) = L$. The achieved error is compromised as much as possible toward the maximum allowed aggregation error ($E_{\max}$) in order to maximize the achieved lifetime. With $h(L,E) = 0.01L - E$, the result falls in between. Both the achieved lifetime and the achieved error are compromised to a certain extent, in order to maximize the objective function.

We make some interesting observations in light of a difference in the pattern of the change of data values between the Melb data sets and the Fisher data set. (As shown in Fig. 8, the value changes quite linearly in Melb, but remains almost constant with occasional spikes in Fisher.) With $h(L,E) = -E$, the pair of achieved lifetime and achieved error is almost the same between Melb and Fisher. This is because the sensors send their readings frequently in order to minimize the aggregation error. With $h(L,E) = L$, the aggregation error is significantly lower for Fisher than for Melb. This is because the filter thresholds of the sensors can be smaller for Fisher, as the magnitude of the change in data values is smaller most of the time (as shown in Fig. 8). With $h(L,E) = 0.01L - E$, the result shows both phenomena to a lesser extent.

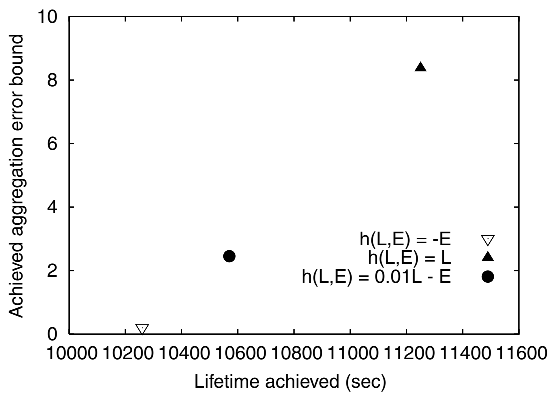
5.4.3. *Comparison with other methods*

To our knowledge, there is no existing method designed to minimize the aggregation error given a lifetime constraint, nor a method designed to optimize a combination of the aggregation error and the lifetime. The only existing methods which can be used for comparison purposes are those that maximize the lifetime given an error constraint [7,19,21]. We have chosen Olson et al.'s [19] method for comparison, because it is designed for a single-hop network and, hence, can be easily adapted to cluster-based routing, while other methods [7,21], based on tree-based routing, cannot. In this comparison, we refer to our method as *QoS* and Olston et al.'s method as Filter. We set up QoS with the same goal as Filter by using the objective function $h(L,E) = L$ and a user-provided constraint $E \leqslant E_{\max}$. Then, we compare the achieved lifetimes (until the first node dies) between QoS and Filter.

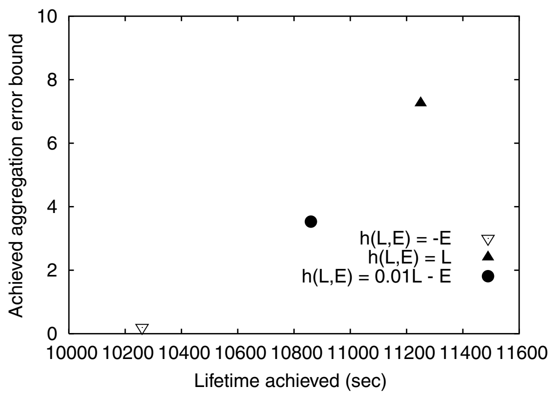
There are three major differences between QoS and Filter. Firstly, QoS optimizes the filter thresholds for a combined lifetime and error goal, whereas Filter does so only to maximize the lifetime while staying within an error bound. In fact, the optimization objective of Filter is only a special case of the optimization objective of QoS. In other words, the objective of QoS is more general, allowing the user to specify in one's goal that "The system can reduce the error by 0.01 for a 1 day reduction in lifetime, while prolonging the lifetime to at least 1 month and keeping the error to 0.5." Secondly, with this difference in the optimization objective, the linear equations QoS generates (at the QP) are different from those Filter generates. As a result, QoS is required to use linear programming to solve the equations, whereas Filter uses its own iterative linear solver. Thirdly, there is also a difference in the data the QP receives. In QoS, the QP receives *T–R* curves (from RPs) whereas in Filter, the QP receives none. Specifically, the QP in Filter only observes the number of readings falling outside the filter threshold (which is continuously shrunken automatically) to determine whether the thresholds should be increased. On the surface, it may seem that QoS is less efficient as it consumes more energy in sending the *T–R* curves. However, as the experiments in this section show, the *T–R* curves help the QP compute so much more optimal filter thresholds that the overall energy usage is generally lower for QoS.

We tried different cases of varying parameters such as those mentioned in the accuracy experiments. The resulting performance graph varies depending on the case, but all the results consistently show that QoS performs better than Filter. This is a very encouraging result, considering that QoS carries the inherent overhead

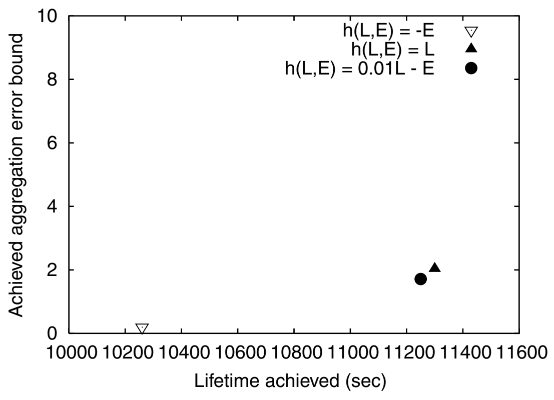
[3] We set $L_{\min}$ lower than 11,200, used in the accuracy experiments, because it increases the optimization search space. A larger search space allows us to see the effect of using different objective functions more clearly.



(a) Melb-max



(b) Melb-min

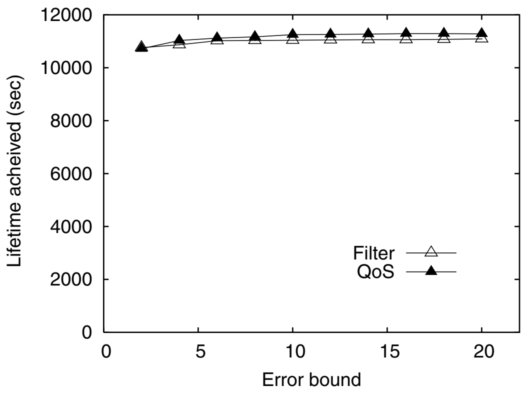


(c) Fisher

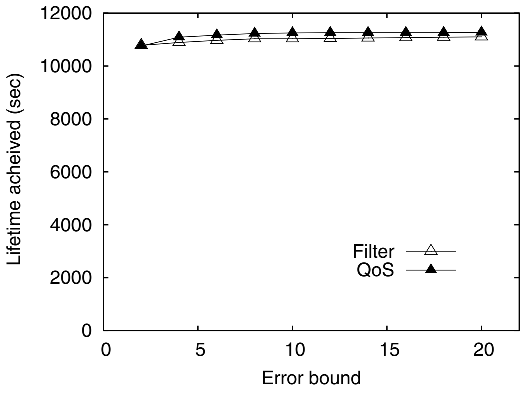
Fig. 10. Optimization computations with different objective functions.

of sending data (i.e., T–R curves, optimized filter thresholds) at each update interval. In Filter, there is no data sent between RPs and the QP (or the sink) at all, except for an instruction from the QP to selected RPs to expand their filter thresholds during the filter adjustment. These results show that, in QoS, the benefit of using T–R curves to compute optimal filter thresholds outweighs the overhead.

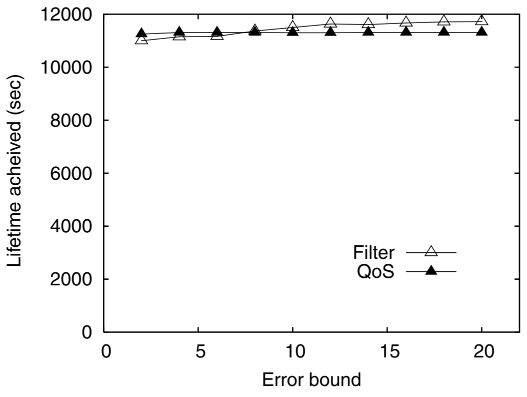
We now present other experiments comparing QoS with Filter and discuss our observations on the results. Fig. 11 shows the result for varying error bound. In both methods, the lifetime increases as the error bound



(a) Melb-max



(b) Melb-min

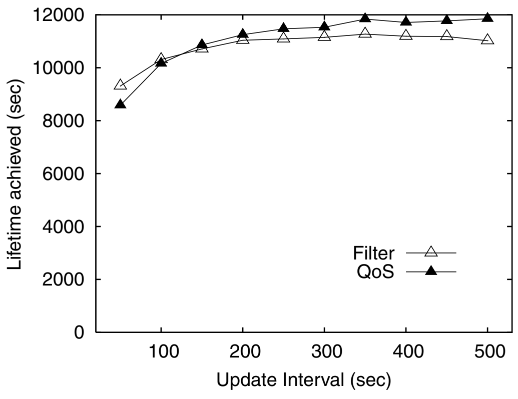


(c) Fisher

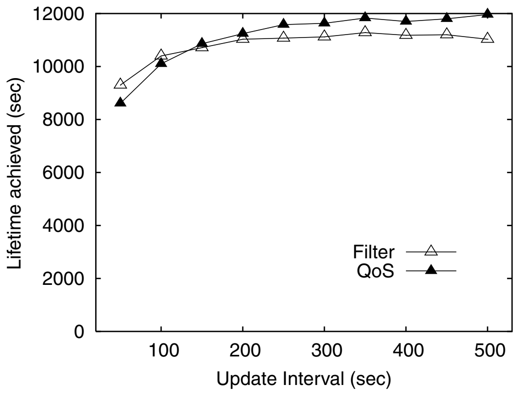
Fig. 11. Achieved lifetime for varying error bound.

increases, which is obvious. The performance of the two methods is very similar for all the error bound values and for all three data sets.

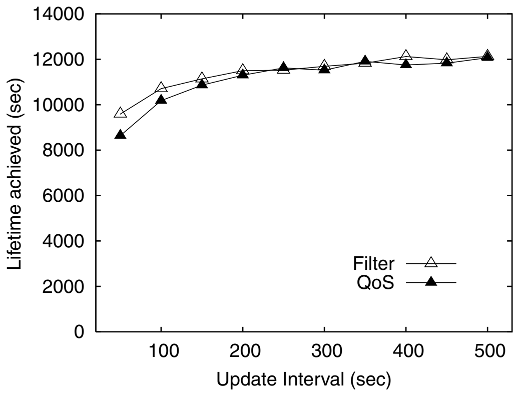
Fig. 12 shows the results of varying the update interval. The update interval is the interval between optimization computations in QoS and the interval between filter threshold expansions in Filter. This figure shows that QoS begins to outperform Filter as the update interval increases. This is due to the fact that, in QoS, a larger update interval leads to less frequent sending of *T*–*R* curves, hence longer lifetime, but in Filter, this



(a) Melb-max



(b) Melb-min



(c) Fisher

Fig. 12. Achieved lifetime for varying update interval.

does not apply since Filter does not send *T–R* curves. It is true that, in Filter, a larger update interval also leads to less frequent sending of an instruction for the filter threshold expansion. However, the resulting increase of achieved lifetime is very small compared with that in QoS. This is because, as mentioned above, there is no data (except for the small amount of instruction) sent by the sink during the update interval in Filter.

Fig. 13 shows the results of varying sensor field area. This figure shows that the lifetimes of both methods decrease as the area increases. This is obvious, since the average distance between nodes increases as the area increases. Sending data across a longer distance increases the frequency of data retransmission and, consequently, the nodes on average consume more battery power.

Fig. 14 shows the results of varying the number of clusters (among the 50 nodes). This figure shows that QoS outperforms Filter when the number of clusters is smaller than 10, but performs approximately the same for a larger number of clusters. This can be explained as follows: when the number of clusters is large, the size of each cluster is small, since the number of nodes is fixed. A smaller cluster means each node acts as the

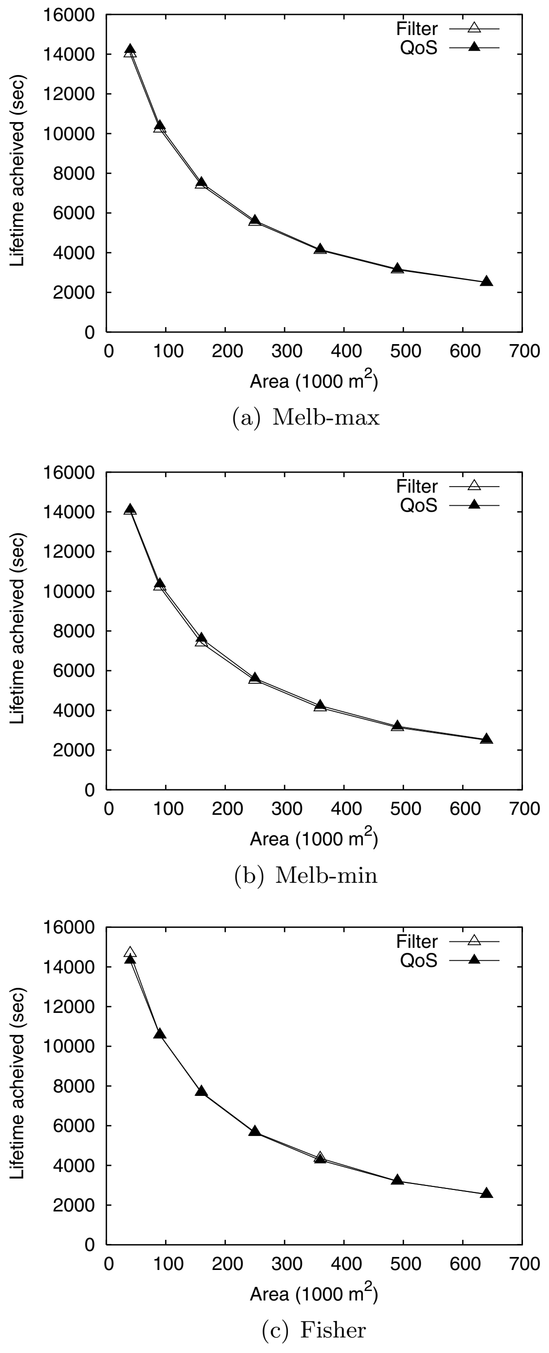
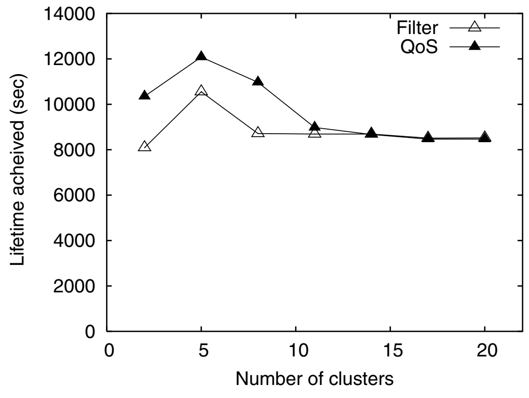
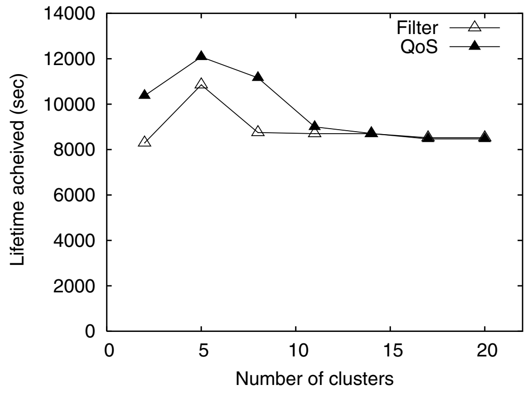


Fig. 13. Achieved lifetime for varying sensor field area.

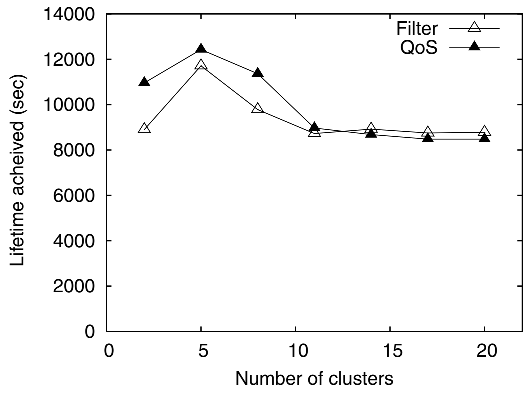
cluster head for a longer duration. When a node acts as a cluster head, it sends the aggregated sample reading to the sink at every sample interval. This is the main cause of energy consumption for cluster heads. Since both methods do the same thing for this part of the normal mode, their performances are approximately the same when the number of clusters is large.



(a) Melb-max



(b) Melb-min



(c) Fisher

Fig. 14. Achieved lifetime for varying number of clusters.

6. Conclusion

Allowing users to specify QoS goals relieves them from the unrealistic expectation of knowing *a priori* the lifetime of the sensor network for a given error tolerance. To achieve user-provided QoS goals, we translated the problem into a mathematical programming formulation. The formulation used in this paper is generic, so it can be customized for different factors such as different network routing models, different configurations for aggregation hierarchy, and so on.

To demonstrate the feasibility of our approach, we instantiated it for a particular system setup and provided concrete mathematical formulations and algorithms for solving the problem. To demonstrate the high accuracy, high flexibility, and low overhead of our solution, we conducted extensive experiments using real data sets on a simulator.

In this paper, we showed the algorithms for SUM only, but other aggregation functions such as COUNT, AVG, MIN and MAX can be supported by modifying the appropriate parts of the optimization equations. We leave this as the future work. Further future work will be to build a probabilistic model into our optimization equations to model varying probability of packet loss and unexpected sensor deaths.

Acknowledgements

We thank the Sensor Networks Work Study Group members at the University of Vermont for their comments on our work. We also thank the anonymous reviewers for their invaluable comments on the original manuscript and thank the Editor-in-Chief Professor Pedrycz for guiding us to improve the linguistic quality of the manuscript. This research has been partly supported by the National Science Foundation under Grant No. IIS-0415023. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Appendix A. Derivations of g and w_{ik}

Here we derive the function $g(L, i)$ and the weight w_{ik} in the system-induced constraints expressed as Eq. (3) of Section 4.3. Tables 2 and 3 respectively show the constants and variables used in the derivation. We assume sensors within a cluster are numbered, and refer to the ith sensor as "sensor i." Additionally, we make the following assumptions on the cluster-based routing. First, given the assumption of synchronized sensor readings (i.e., sampling), we assume that time duration is measured as sampling ticks, i.e., the number of readings made. Second, as all sensors are equally likely to be selected as a cluster head, we assume that all the sensors in the same cluster take turns to be selected once and only once in each round of turns.

Given these assumptions, for each sensor i in the network lasting lifetime L, R_i (i.e., the fraction of the readings sent by the sensor) is expressed as follows:

$$R_i = \frac{B_i - B_{\mathrm{h}_i}(L) - B_{\mathrm{n}_i}(L)}{B_{\mathrm{s}_i} L} \tag{6}$$

Table 2
Constants used to derive the expressions of $g(L, i)$ and w_{ik} in Eq. (3)

Symbol	Description
B_i	Battery power currently left in sensor i
B_{s_i}	Average battery power consumed by sensor i to send one reading (to its cluster head)
$B_{\mathrm{r}_{ik}}$	Battery power consumed by sensor i to receive a reading from sensor k
$C(i)$	Set of sensors in the same cluster as, but excluding, sensor i
D_i	Duration of a sensor i being a cluster head each time it is selected
N_i	Number of sensors in the cluster to which sensor i belongs
P_i	Duration for one round of turns of selecting a cluster head in the cluster to which sensor i belongs. ($P_i = D_i N_i$)
O_{h_i}	Overhead on a sensor i while being a cluster head during D_i
O_{n_i}	Overhead on a sensor i while not being a cluster head during D_i

Table 3
Variables used to derive the expressions of $g(L, i)$ and w_{ik} in Eq. (3)

Symbol	Description
R_i	Fraction of the number of readings sent (to the cluster head) over the number of readings sampled by sensor i
L	Lifetime of the sensor network, measured as the number of sampling ticks
R_{ik}	Fraction of readings sent to sensor i by sensor k that belongs to cluster $C(i)$
$B_{\mathrm{h}_i}(L)$	Battery power consumed by a sensor i while being a cluster head in a network lasting lifetime L
$B_{\mathrm{n}_i}(L)$	Battery power consumed by a sensor i while not being a cluster head in a network lasting lifetime L

The reasoning behind this equation is as follows: the numerator is the battery power that would remain available for sensor i to send readings after consuming the battery power $B_{\mathrm{h}_i}(L)$ while being a cluster head and the battery power ($B_{\mathrm{n}_i}(L)$) while not being a cluster head. Here, the latter is due to an overhead inherent in a sensor, even when no reading is sent or received. Then, the numerator divided by B_{s_i} is the number of readings sent (to a cluster head) during the lifetime. (Note that, while a sensor is a cluster head, its readings are not sent to a cluster head (i.e., itself) and, therefore, B_{s_i} is the *average* power consumed per sending.) Since time duration is measured as sampling ticks, L is the total number of samples that would be taken during the lifetime (clock time) of the sensor network. Hence, further dividing the result (of division by B_{s_i}) by L results in the fraction R_i during the lifetime.

Next, the two variables in Eq. (6), $B_{\mathrm{h}_i}(L)$ and $B_{\mathrm{n}_i}(L)$, are expressed as:

$$B_{\mathrm{h}_i}(L) = \frac{L}{P_i}\left(D_i\left(\sum_{k\in C(i)} R_{ik}B_{\mathrm{r}_{ik}}\right) + O_{\mathrm{h}_i}\right) \tag{7}$$

$$B_{\mathrm{n}_i}(L) = \frac{(N_i - 1)L}{P_i}O_{\mathrm{n}_i} \tag{8}$$

The reasoning behind Eq. (7) is as follows: during the time sensor i is the cluster head, for each sensor k in the cluster $C(i)$, the fraction of readings sent to sensor i is R_{ik} (see Section 4.3). Thus, the number of readings received by sensor i from sensor k each time it is a cluster head is D_iR_{ik}, and the battery power consumed by sensor i to receive that many readings is $D_iR_{ik}B_{\mathrm{r}_{ik}}$. Considering all sensors in $C(i)$, we sum the above term for all $k \in C(i)$. Then, we add the overhead of being a cluster head O_{h_i} and multiply the result by the number of times sensor i is a cluster head, i.e., the number of the rounds of cluster head turns (L/P_i). The reasoning behind Eq. (8) is as follows: sensor i is *not* selected as a cluster head $N_i - 1$ times during P_i (for one round of turns). Since there are L/P_i rounds of turns during the sensor networks' lifetime L, the number of times sensor i is not a cluster head during L equals $(N_i - 1)L/P_i$. By multiplying this to the overhead O_{n_i}, we obtain Eq. (8).

Now, by substituting Eqs. (7) and (8) into Eq. (6), we obtain the following expressions for $g(L, i)$ and w_{ik}.

$$g(L,i) = \frac{B_i}{B_{\mathrm{s}_i}L} - \frac{O_{\mathrm{h}_i}}{B_{\mathrm{s}_i}P_i} - \frac{(N_i - 1)O_{\mathrm{n}_i}}{B_{\mathrm{s}_i}P_i} \tag{9}$$

$$w_{ik} = \frac{B_{\mathrm{r}_{ik}}}{B_{\mathrm{s}_i}N_i} \tag{10}$$

References

[1] A. Amis, R. Prakash, T. Vuong, D. Huynh, Max–min *d*-cluster formation in wireless ad hoc networks, in: Proceedings of IEEE Conference on Computer Communications (INFOCOM), 2002, pp. 32–41.

[2] A.D. Amis, R. Prakash, Load-balancing clusters in wireless ad hoc networks, in: Proceedings of IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET), 2000, p. 25.

[3] S. Bandyopadhyay, E.J. Coyle, An energy efficient hierarchical clustering algorithm for wireless sensor networks, in: Proceedings of IEEE Conference on Computer Communications (INFOCOM), 2003, pp. 1713–1723.

[4] M. Chatterjee, S.K. Das, D. Turgut, WCA: A weighted clustering algorithm for mobile ad hoc networks, Cluster Computing 5 (2) (2002) 193–204.

[5] G. Chen, J. Branch, M. Pflug, L. Zhu, B. Szymanski, Advances in Pervasive Computing and Networking, Springer, 2005.

[6] J. Considine, F. Li, G. Kollios, J. Byers, Approximate aggregation techniques for sensor databases, in: Proceedings of IEEE International Conference on Data Engineering (ICDE), 2004, pp. 449–460.
[7] A. Deligiannakis, Y. Kotidis, N. Roussopoulos, Hierarchical in-network data aggregation with quality guarantees, in: Proceedings of International Conference on Extending Database Technology (EDBT), 2004, pp. 658–675.
[8] K.-W. Fan, S. Liu, P. Sinha, Structure-free data aggregation in sensor networks, IEEE Transactions on Mobile Computing 6 (8) (2007) 929–942.
[9] M.J. Handy, M. Hasse, D. Timmermann, Low energy adaptive clustering hierarchy with deterministic cluster-head selection, in: Proceedings of International Workshop on Mobile and Wireless Communications Network (MWCN), 2002, pp. 368–372.
[10] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks, in: Proceedings of Hawaii International Conference on System Sciences (HICSS), 2000, p. 8020.
[11] K.W. Hipel, A.I. McLeod, Time Series Modelling of Water Resources and Environmental Systems, Elsevier, 1994.
[12] K. Kalakis, V. Puttagunta, P. Namjoshi, Accuracy vs. lifetime: Linear sketches for approximate aggregate range queries in sensor networks, Technical Report 4, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore County, 2004.
[13] K. Kalpakis, K. Dasgupta, P. Namjoshi, Maximum lifetime data gathering and aggregation in wireless sensor networks, in: Proceedings of IEEE Conference on Networks (ICN), 2002, pp. 685–696.
[14] S. Madden, M.J. Franklin, J.M. Hellerstein, TAG: a tiny aggregation service for ad-hoc sensor networks, in: Proceedings of Symposium on Operating Systems Design and Implementation (OSDI), 2002, pp. 131–146.
[15] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TinyDB: an acquisitional query processing system for sensor networks, ACM Transactions on Database Systems 30 (1) (2005) 122–173.
[16] A. Manjhi, S. Nath, P.B. Gibbons, Tributaries and deltas: efficient and robust aggregation in sensor network streams, in: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), 2005, pp. 287–298.
[17] A.B. McDonald, T. Znati, A mobility based framework for adaptive clustering wireless ad-hoc networks, IEEE Journal on Selected Areas in Communications 17 (8) (1999) 1466–1487.
[18] S. Nath, P.B. Gibbons, S. Seshan, Z.R. Anderson, Synopsis diffusion for robust aggrgation in sensor networks, in: Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys), 2004, pp. 250–262.
[19] C. Olston, J. Jiang, J. Widom, Adaptive filters for continuous queries over distributed data streams, in: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), 2003, pp. 563–574.
[20] Q. Ren, Q. Liang, Energy and quality aware query processing in wireless sensor database systems, Information Sciences 177 (10) (2007) 2188–2205.
[21] M.A. Sharaf, J. Beaver, A. Labrinidis, P.K. Chrysanthis, Balancing energy efficiency and quality of aggregate data in sensor networks, VLDB Journal 13 (4) (2004) 384–403.
[22] M. Sharifzadeh, C. Shahabi, Supporting spatial aggregation in sensor network databases, in: Proceedings of ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS), 2004, pp. 166–175.
[23] H.O. Tan, I. Korpeoglu, Power efficient data gathering and aggregation in wireless sensor networks, SIGMOD Record 32 (4) (2003) 66–71.
[24] Y. Yao, J. Gherke, Query processing for sensor networks, in: Proceedings of International Conference on Innovative Databse Systems Research (CIDR), 2003, p. 21.
[25] M. Younis, M. Youssef, K. Arisha, Energy-aware routing in cluster-based sensor networks, in: Proceedings of IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS), 2002, pp. 129–136.
[26] O. Younis, S. Fahmy, Distributed clustering in ad-hoc sensor networks: a hybrid, energy-efficient approach, in: Proceedings of IEEE Conference on Computer Communications (INFOCOM), page 13_5, 2004.
[27] J. Zh, S. Papavassiliou, J. Yang, Adaptive localized qos-constrained data aggregation and processing in distributed sensor networks, IEEE Transactions on Parallel and Distributed Systems 19 (9) (2006) 923–933.
[28] J. Zhu, S. Papavassiliou, S. Kafetzoglou, J. Yang. An efficient QoS-constrained data aggregation and processing approach in distributed wireless sensor networks, in: Proceedings of IEEE Symposium on Computers and Communications (ISCC), 2006, pp. 257–262.
[29] J. Zhu, S. Papavassiliou, S. Kafetzoglou, J. Yang. On the modeling of data aggregation and report delivery in QoS-constrained sensor networks, in: Proceedings of IEEE Pervasive Computing and Communications Workshops (PerCom), 2006, pp. 347–351.