The VLDB Journal
DOI 10.1007/s00778-006-0016-z

REGULAR PAPER

Bandwidth-constrained queries in sensor networks

Antonios Deligiannakis · Yannis Kotidis · Nick Roussopoulos

Received: 4 March 2005 / Accepted: 3 December 2005
© Springer-Verlag 2006

Abstract Sensor networks consist of battery-powered wireless devices that are required to operate unattended for long periods of time. Thus, reducing energy drain is of utmost importance when designing algorithms and applications for such networks. Aggregate queries are often used by monitoring applications to assess the status of the network and detect abnormal behavior. Since radio transmission often constitutes the biggest factor of energy drain in a node, in this paper we propose novel algorithms for the evaluation of *bandwidth-constrained queries* over sensor networks. The goal of our techniques is, given a target bandwidth utilization factor, to program the sensor nodes in a way that seeks to maximize the accuracy of the produced query results at the monitoring node, while always providing strong error guarantees to the monitoring application. This is a distinct difference of our framework from previous techniques that only provide probabilistic guarantees on the accuracy of the query result. Our algorithms are equally applicable when the nodes have ample power resources, but bandwidth consumption needs to be minimized, for instance in densely distributed networks, to ensure proper operation of the nodes. Our experiments with real sensor data show that bandwidth-constrained queries can substantially reduce the number of messages in the network while providing very tight error bounds on the query result.

Keywords Sensor networks · In-network aggregation

A. Deligiannakis (✉)
University of Athens, Athens, Greece
e-mail: adeli@di.uoa.gr

Y. Kotidis
Athens University of Economics and Business,
Athens, Greece
e-mail: kotidis@aueb.gr

N. Roussopoulos
University of Maryland, College park, USA
e-mail: nick@cs.umd.edu

1 Introduction

Sensor networks consist of hundreds or thousands of inexpensive wireless sensor nodes. Their applications span different domains, from habitat monitoring to location tracking and inventory management. Each sensor node is able to sense its surroundings by obtaining appropriate measurements, perform simple computations and relay the results to other nodes in the network. Processing in sensor networks is often driven by designated monitoring nodes, which usually possess increased processing, storage and energy resources, when compared to the other nodes in the network. These monitoring nodes often evaluate the current state of the network by issuing *continuous queries* [5,37] over the data collected by the sensors.

Because of the multi-hop communication between nodes in sensor networks, the broadcast nature of the transmitted messages and the high density of nodes in a typical installation, collecting individual node measurements at the monitoring node is immensely expensive. Aggregation is an effective means to reduce the data measurements into a much smaller set of comprehensive statistics, like sum, average etc. In order to obtain the full benefits of data aggregation, recent proposals perform the process inside the network [4,19,24,34]. First, a routing path, which is commonly referred to

as the *aggregation tree*, to the monitoring node is established. Then, through the use of a carefully designed transmission schedule, nodes are programmed to combine measurements that they receive from their descendants in the topology and propagate a single value to their ancestors. With proper synchronization [24], the number of messages required to update the aggregate at the monitoring node is equal to the number of edges in the aggregation tree.

In-network data aggregation has been shown to reduce the number of messages in the network, often by more than an order of magnitude [24,32]. However, in large networks, especially when the monitoring node is several *wireless hops* away, the cost of aggregation may still be significant. In densely distributed networks, proper control on the bandwidth consumed by each running query is essential to guarantee that parts of the network are not overburdened and that all required processing can be performed by the network. For instance, a continuous user query that aggressively computes the average temperature readings of all sensors nodes every second may consume all available bandwidth and essentially block out other significant processing assigned to the nodes. As our experimental evaluation demonstrates, often a substantial reduction in the consumed bandwidth during aggregate computation, achieved by suppressing some update messages, only slightly impacts the accuracy of the produced results.

Having control over the bandwidth consumed by a posed aggregate query is also important to ensure the longevity of the network, since transmission is the biggest source of energy drain in sensor nodes [14]. This may not be a major concern when sensor nodes are attached to larger devices with ample power supply, but becomes critical when nodes are powered by small batteries. The reduction in bandwidth consumption results in an equally important reduction in the energy consumption of the sensor nodes, since this is often directly proportional to the number of transmitted and received bits [17,23,36]. We need to note that, depending on the radio technology used, each transmitted message may drain energy not only from the transmitting node and the intended recipient, but also from other nodes in the vicinity of the transmitting node which also receive the message due to the wireless nature of the communication.

1.1 Limitations of prior techniques

A straightforward method to limit the bandwidth consumption of a query would be to increase the sampling period (often called *epoch duration*[24]) of collecting a new measurement. However, such an approach leaves us unable to observe and react quickly to changes in the behavior of nodes within the longer epoch duration. Recently, novel algorithms that build probabilistic models of the observed data and then use these models to probe the sensors for their measurements in a limited amount of epochs, depending on the confidence of the constructed model, have been proposed [13,22]. However, these techniques cannot provide strong deterministic guarantees on the quality of the produced result. The provided guarantees are only probabilistic ones, and the accuracy of these guarantees depends on whether the real-time observed data are similar to the training data used for building the model. As the authors of [13] state, "for models to perform accurate predictions they must be trained in the kind of environment where they will be used". While the acquisition of the appropriate set of training data may be feasible in scenarios of controlled environments, such as temperature monitoring applications within a lab, in applications where the sensors are thrown over hostile environments or disaster areas this is an unrealistic assumption. Moreover, the techniques in [13] cannot be used for the detection of outliers, meaning events or measurements that deviate significantly from either the corresponding values observed in other sensors, or in the same sensor but over prior time periods. However, the purpose of *monitoring* applications is often to detect such large deviations from the normal behavior, since these deviations may trigger some alert or require appropriate action to be taken. Alternative techniques that perform some sort of sampling (uniform or biased) of the data sources face similar drawbacks.

1.2 Our proposal

A more robust approach to limit the bandwidth consumption of a query is to let the application define the required epoch duration and a bandwidth utilization factor B_Util. This is a single parameter with a value between 0 and 1. A value of 1 for B_Util corresponds to an unrestricted evaluation of the query, where each node makes one transmission per epoch. This results in an exact computation of the aggregate at the monitoring node using T transmissions, where T is the number of nodes used in the query, besides the monitoring node. A value of B_Util lower than 1 corresponds to a *bandwidth constrained* query evaluation, during which the average number of transmissions per epoch will be $B_Util \times T$.

In a *bandwidth constrained* query evaluation, we would like the aggregate value $\hat{V}$ reported to the monitoring node to be as close as possible to the true aggregate value V of the measurements collected by the sensor nodes. In order to provide strong deterministic

guarantees, the objective of our algorithm will be to minimize the maximum possible error $|V - \hat{V}|$ and make it available at the monitoring node. The existence of these error guarantees at each time epoch, independently of the characteristics of the monitored data, is a distinct difference of our approach from prior probabilistic techniques.

A premise when designing algorithms for sensor network applications is that they should require minimum state information and that this information should be readily available in each node. Collecting *individual* node statistics at a central node for (re-)programming the nodes is not efficient because of the overhead for communicating this information, which cannot be aggregated since individual node statistics are required. In our work we will, thus, seek to develop an algorithm that bases its decisions on a small number of easy to calculate statistics that can be aggregated during query processing.

In this paper we propose the *marginal gains adjustment* (MGA) algorithm, a localized algorithm based on the idea of marginal gains, for the problem of bandwidth-constrained aggregate continuous queries over sensor networks. Similarly to prior approaches [10,27,32] that seeked to reduce the bandwidth consumption in sensor networks, our MGA algorithm initially installs an error filter in each node. What is important though is the way that, depending on the actual bandwidth consumption, our MGA algorithm periodically modifies these filters in order to try to equate the actual bandwidth consumption to the desired one, but also to provide as strict error guarantees as possible. These error guarantees are always known to the monitoring node.

1.3 Contributions

Our contributions are summarized as follows:

1. We introduce the notion of bandwidth-constrained queries as a means of reducing the bandwidth consumption and energy drain during query evaluation, while providing deterministic error guarantees to the monitoring application. We illustrate how the reduction in the number of messages transmitted during query execution translates to reduced energy drain in the network using a simplified cost model. This allows the application to properly modify the bandwidth alloted to a continuous query based on the desired average energy consumption in the network.
2. We introduce the notion of the *marginal gain* of a node or an entire subtree and use it in our algorithm to determine the amount of bandwidth allocated to each node or subtree. The operation of our algorithm is symmetric for the cases of both possible under-utilization and over-utilization of the bandwidth. The algorithm proceeds by disseminating positive or negative bandwidth amounts in a localized manner between parent and child nodes in the aggregation tree. Each node then translates the received bandwidth amount to an appropriate decrease or increase, respectively, of its filter width. This procedure is performed using simple, local statistics maintained by the nodes during the execution of the query.
3. While our algorithm is designed to limit the average bandwidth consumption during query execution, we also discuss simple extensions that allow us to provide strict bandwidth guarantees to all or parts of the aggregation tree. These extensions are often necessary to ensure that individual areas of the network are not over-burdened during query execution. We also discuss the necessary extensions in case the aggregation tree is modified, for example when nodes join or leave the query.
4. We present adaptations of prior techniques for the evaluation of bandwidth-constrained queries in sensor networks and draw direct comparisons to our MGA algorithm.
5. We present an extensive experimental analysis of our algorithm in comparison to previous techniques using real and synthetic sensor data. Our experiments validate our approach and demonstrate that, for the same average bandwidth consumption of the query, our techniques outperform previous techniques by providing significantly more tight, often by more than an order of magnitude, error guarantees. In one case, using a bandwidth constrained query we achieved a relative error in the approximation of just 0.1% using 5% of the bandwidth that an unconstrained evaluation would require. Thus, many applications can benefit from our framework by substantially reducing the number of messages, with little sacrifice in the quality of the aggregates.

The rest of the paper is organized as follows. Section 2 presents related work. In Sect. 3 we comment on the sources of energy drain in sensor nodes and show that bandwidth constrained queries can have a significant impact in the network's lifetime. In Sect. 4 we detail our problem and present our framework. In Sect. 5 we present our MGA algorithm for dynamically programming the sensor nodes to operate in bandwidth-constrained queries and also discuss interesting extensions to the original algorithm. In Sect. 6 we present alternative techniques for our application, discuss

their advantages and shortcomings, and draw direct comparisons to our method. Section 7 contains our experiments, while Sect. 8 contains concluding remarks.

2 Related work

There have been several recent proposals, such as COUGAR [38] and TinyDB [24], on using embedded database systems in sensor networks. A declarative query language like SQL provides far greater flexibility than hand-coded programs that are pre-installed at the sensor nodes [25]. Another advantage is that the database system can be used to provide energy-based query optimization. In [24], nodes of the aggregation tree carefully synchronize the periods when they transmit data. The idea is to subdivide each epoch into intervals and have parent nodes in the tree listen for messages from their children during pre-defined time-slots. This allows the nodes to power-down their radios when not necessary and reduce their energy and bandwidth consumption. Another notable method for synchronizing the transmission periods of nodes is the recently proposed wave scheduling approach of [12]. These optimizations are orthogonal to our framework. In fact, our algorithm should be implemented on top of a protocol like TAG [24] to obtain its maximum benefits.

The networking aspects of wireless sensor nodes is a topic that has gained a lot of attention in the networking community. Because of the unattended nature of these networks, nodes must co-operate to perform the task at hand. Thus, nodes must be able to self-configure [3], discover their surrounding nodes [14,16] and compute energy-efficient data routing paths (such as the aggregation tree [19]) [4,17,23,24,34,36]. The techniques developed in the above work are complementary to our work, since while the above techniques help determine energy-efficient aggregation trees, our algorithms further reduce the amount of information flowing in the network. In [8], a framework for compensating for packet loss and node failures during query evaluation is proposed. In the database community additional issues such as data modeling and acquisition [13,21] and data compression [9,11] have been recently addressed. Distributed storage management is another topic that brings together the networking and database communities [12,15,31].

Our work is also related to the area of continuous queries over data streams, which has been broadly studied in recent years [5,18,26]. Olston et al. [28–30] investigated the tradeoffs between precision and performance in cached and replicated data. In [30] the emphasis is on determining when cached objects of remote data sources should be refreshed in order to minimize the average divergence of the cached objects given a server-size bandwidth constraint. The used divergence function can either represent the staleness or the update lag of the cached object, or the value deviation between the object's cached and current values. In our experiments (Sect. 7) we present a modification of this algorithm, that we term as the *threshold-based adjustment* algorithm (TBA), for the case of sensor networks and compare its performance to our MGA algorithm. The work in [27] also discusses extensions for the execution of multiple concurrent continuous queries. Several of these ideas can also be combined with our algorithms. On the other hand, earlier work in distributed constraint checking [1,35] cannot be directly applied in our setting, because of the different communication model and the limited resources at the sensors. The evaluation of probabilistic queries over imprecise data was studied in [6,7]. Extending this work to hierarchical topologies, such as the ones studied in our paper, is an open research topic. Finally, [2,20] investigate decentralized algorithms for aggregate computations with applications in P2P and sensor networks.

The problem of minimizing the number of messages exchanged in the network for a given error constraint has been studied in [10,32]. This is the dual problem of the one that we consider in this paper. In [32] the authors suggest using a uniform distribution of the error, while [10] distributes the error based on local statistics collected on a node. The problem that we study here is harder than the dual worked in [10] for hierarchical networks and in [27] for flat topologies. In the dual problem, both the given constraint and the quantity controlled at each node are the same; namely the error of either the application or of individual nodes. Moreover, in the dual problem the overall error of the application cannot change based on the underlying data distribution, and no guarantees are given for the bandwidth consumption. On the contrary, in the evaluation of bandwidth-constrained queries, the given constraint is the bandwidth, which is different than the quantity that we control on each node: the node's filter setup. As is explained in Sect. 5, the overall bandwidth consumption needs to be carefully monitored and then properly disseminated to nodes and *translated* in them to update their error filters.

While in [10] all nodes have the same behavior during updates (they all shrink their filters to generate the error budget, as was suggested in [27]), in our algorithm a node may shrink or expand its filter. This decision depends on the observed bandwidth consumption, its atomic behavior in the past few epochs and the computed marginal gains of a change in its filter. Moreover, note that the techniques in [10] possess two main drawbacks when

applied to bandwidth-constrained queries. In [10] even when the data characteristics do not vary and a good filter configuration has been reached, a considerable amount of messages needs to be transmitted in each update period due to the constant modification of the filters. This makes it practically impossible to achieve small bandwidth targets. On the contrary, in our case if the bandwidth consumption is close to the target no update messages are sent. Moreover, the error dissemination process in [10] considers only the potential gain of nodes/subtrees and not the size of the error filters, thus making the process biased towards nodes with large filters. The algorithms in this paper do not suffer from the same drawback, thus allowing for the judicial allocation of bandwidth on each sensor node.

3 Benefits of constrained bandwidth utilization

In this section we provide some background information needed in the discussion of this paper. We first present a description of the characteristics of sensor nodes, focusing on the sources of energy drain in sensor networks, and then provide an analysis of the expected benefits on the network's lifetime when applying techniques that constrain the bandwidth consumption in these networks.

3.1 Characteristics of sensor nodes

Each sensor node may be composed of several parts including, among others, a processing unit (CPU), a memory component, a battery that supplies the sensor with the required energy for its operation, a radio transmitter that facilitates the communication with other sensor nodes, and any sensing elements required for the collection of the sensor measurements. Examples of such sensing elements include microphones for acoustic sensing, accelerometers and temperature sensors.

Most sensor network applications face severe energy and bandwidth constraints. The deployment of the sensor nodes is typically unattended, and replacing the sensor batteries may not only be expensive, but sometimes impossible to do right away (e.g. disaster areas). Moreover, while the processing and memory capabilities of sensor nodes increase at a rate similar to Moore's law, the amount of energy stored in the batteries used in such nodes has exhibited a mere 2–3% annual growth. On the other hand, using sensor nodes with large battery units, and therefore more energy, is often not a viable solution due to the associated increased size and cost of the sensor nodes. Therefore, unless the sensors are attached to and powered by a larger unit, designing energy efficient protocols is essential to increase the lifetime of the sensor network.

The actual energy consumption by each sensor node depends on its current state. In general, each sensor node can be in one of the following states:

1. Low-duty cycle: In this state the node is at a sleep mode, consuming a minimal amount of energy. The sensor node performs no processing or communication during this state.
2. Receiving/Transmitting/Idle Listening: In this state the node either receives, transmits or listens for data or control messages intended for the node. The cost of transmission increases rapidly with its *range*, that is the maximum distance from the transmitting node within which other sensors are able to receive the message. For long-distance radios, the transmission cost dominates the receiving and idle listening costs. For short-range radios, these costs are comparable. For example, in the Berkeley MICA2 motes (Fig. 1), the power consumption ratio of transmitting/receiving at 433 MHz with RF signal power of 1mW is 1.41:1 [39]. This ratio becomes larger than 3:1 when the radio transmission power is increased [33]. The idle listening cost is substantial and often comparable to the energy of receiving data.
3. Processing: The node performs computation based on its obtained measurements and its received data. The cost of processing can be significant but is generally much lower than the cost of transmission. For example, in the Berkeley MICA nodes sending one bit of data costs as much energy as 1,000 CPU instructions [25].

The actual energy consumption in any of the above states depends on the actual model of the sensor node and its radio and CPU characteristics. Table 1 summarizes some of the characteristics for the MICA2 mote. To increase the lifetime of the network, some common goals of sensor network applications are (in order of



Fig. 1 The MICA2 Mote

Table 1 Characteristics of the MICA2 mote

Characteristic	Value
CPU	7.3828 MHz
Memory	4 KB SRAM, 128 KB FLASH
Additional storage	32 KB EEPROM
Transmission range	1000 ft
Battery	2 AA Batteries
Radio current draw	25 mA (Transmission max power) 8 mA (receiving) <1 uA (sleep)

importance) to maximize the time when a node is in a low-duty cycle, to reduce the amount of transmitted and received data, to reduce the idle listening time and to reduce processing.

3.2 Energy benefits of bandwidth-constrained queries

We now seek to evaluate how bandwidth-constrained queries reduce the energy drain in sensor nodes. We construct a simple model that estimates the current total energy of the nodes participating in the evaluation of the continuous query. The calculations are based on the number of transmitted and received messages and the corresponding average transmission and reception costs per message (E_{T} and E_{R}, respectively). Also, let E_{I} denote the cost of idle listening, which occurs when a sensor node listens to its channel awaiting for messages. The transmission cost E_{T} incorporates any additional energy consumed in order to reserve the channel for the transmission of a message, or to send/receive acknowledgments for successfully receiving each message. For simplicity, we do not take into account the computational costs, due to the small power consumption required to perform the simple aggregation step in each node (see Sect. 3.1).

Let $E_{\mathrm{T}} = k \times E_{\mathrm{R}}$, where k denotes the ratio between the average energy consumed during the transmission of a message and the corresponding energy consumed while receiving a message. Also, let $E_{\mathrm{R}} = p \times E_{\mathrm{I}}$, where p denotes the ratio between the average energy consumed while receiving a message and the corresponding energy consumed during idle listening.

Consider, for simplicity, that in our model each non-leaf node has *on average* f children nodes. Since all the nodes in the aggregation tree besides the monitoring node are the children of some other node, we can extract the following relationship between the total number of nodes T and the number of non-leaf nodes $T_{\mathrm{non-leaf}}$:

$$T_{\mathrm{non-leaf}} \times f = T_{\mathrm{leaf}} + T_{\mathrm{non-leaf}} - 1 = T - 1.$$

Using a protocol like TAG [24], in an unconstrained query evaluation each non-leaf node needs to keep its radio open during an epoch for enough time in order to receive the f messages from its children nodes containing the updated value of the partial aggregate corresponding to their subtrees (see the following sections for details). This listening cost is not incurred by the leaf nodes of the tree. In our model, the total cost of receiving will thus be exactly:

$$RC_{\mathrm{uncon}} = T_{\mathrm{non-leaf}} \times (f \times E_{\mathrm{R}}) = (T-1) \times E_{\mathrm{R}}$$

while the cost of transmitting the aggregate values will be equal to (the *Monitor* node does not make a transmission):

$$TC_{\mathrm{uncon}} = (T-1) \times E_{\mathrm{T}} = k \times RC_{\mathrm{uncon}}.$$

A protocol implemented on top of TAG that restricts the bandwidth consumed for the execution of the continuous aggregate query to a fraction B_Util of the bandwidth required by an unconstrained evaluation, will result in $1-B_Util$ of the messages not being transmitted and the transmission energy drain will be:

$$TC_{\mathrm{con}} = (T-1) \times B_Util \times E_{\mathrm{T}} = B_Util \times k \times RC_{\mathrm{uncon}}.$$

The total energy drain spent when either receiving packets or performing idle listening will be:

$$\begin{aligned} RC_{\mathrm{con}} &= T_{\mathrm{non-leaf}} \times f \times (B_Util \times E_{\mathrm{R}} + E_I(1-B_Util)) \\ &= (T-1) \times E_{\mathrm{R}} \times (B_Util + (1-B_Util)/p) \\ &= (B_Util + (1-B_Util)/p) \times RC_{\mathrm{uncon}}. \end{aligned}$$

If we consider the time needed for network's energy to reach a value y, when the total initial energy was $C(t_0)$, the unconstrained evaluation will require

$$t_{\mathrm{uncon}} = \frac{C(t_0) - y}{TC_{\mathrm{uncon}} + RC_{\mathrm{uncon}}} = \frac{C(t_0) - y}{RC_{\mathrm{uncon}}(1+k)}$$

epochs, while the bandwidth-constrained execution will reach this energy level after

$$t_{\mathrm{con}} = \frac{C(t_0) - y}{RC_{\mathrm{uncon}}((1+k-1/p) \times B_Util + 1/p)}$$

epochs. Therefore, for any total initial energy level $C(t_0)$, the bandwidth-constrained execution will reach this level y after

$$\frac{1+k}{(1+k-1/p) \times B_Util + 1/p}$$

times more epochs than an unconstrained query execution.

Of course, this is an oversimplified calculation, assuming that all nodes have sufficient initial energy levels for the duration of the query and that the aggregation tree is not modified. In Sect. 5.4 we discuss extensions of

our algorithm in the case of areas of the network with strict bandwidth constraints, nodes with severe energy constrains and node movement.

Example 1. Assuming use of MICA2 nodes at their default and maximum transmission powers (Table 1) we get $k = 1.41$ and 3, respectively (ignoring the cost of control messages, retransmissions and acknowledgments), while $p = 1$. For $B_Util = 6\%$, a bandwidth constraint query evaluation will reach the same energy level in 2.25 and 3.48 times, respectively, more epochs. However, as our experimental evaluation demonstrates, the aggregate computation during this constrained query execution may often incur only a small error (i.e., only 0.1% relative error in Table 7).

Depending on the type of protocol (scheduled or contention [39]) being used by the sensor nodes, each transmitted message may have many hidden costs associated with it. In many protocols (i.e., the CSMA/CA protocol), prior to the transmission of each message some control messages (RTS/CTS) need to be exchanged in order for the transmitting node to gain access to the channel. Similarly, acknowledgments for each received packet often need to be sent. Collisions between transmitted messages may occur and retransmissions (which may result in a significant energy waste) are necessary. By reducing the number of transmitted messages, the number of transmitted control messages and the probability of collisions occurring are greatly reduced. Note that all these costs (including the corresponding receiving costs for all these messages) have been incorporated into the E_T parameter in our model, and thus the actual k value is significantly larger (often by more than a factor of 5) than simply the power ratio between the transmit and receive modes used in the example above. Finally, we note that in a constrained query execution, each node that decides not to transmit a message may immediately go into a low-duty cycle state and, thus, preserve large amounts of energy.

A point worth mentioning is that our algorithms and analysis focuses on the average energy drain on the network without concern on individual nodes whereabouts. The extensions that we discuss in Sect. 5 allow our techniques to provide for nodes that face severe energy constraints. However, it is our belief that applications are better not utilizing strict controls on the operations of individual nodes. Sensor networks often contain redundant nodes to ensure, for instance, coverage on regions with non-uniform communication density and cope with unexpected node failures. Thus, our focus should not be on extending the lifetime of individual nodes but on ensuring that the network *as a whole* has enough resources to perform the task at hand. Prior work on sensor networks (for instance [3,14,17,21]) has built upon these ideas and has been our inspiration for focusing instead on the average energy drain among all nodes in the aggregation tree.

4 A Framework for evaluation of bandwidth-constrained queries

Problem definition and setup. We consider the case of bandwidth-constrained aggregate continuous queries in error-tolerant monitoring applications. In our discussion we focus on queries containing the SUM aggregate function. Weighted SUM aggregates can be computed in an identical way, since in our framework the error filter installed at each node is applied over the (weighted) partial aggregate computed over the values observed in the node's subtree. The COUNT function can always be computed exactly as the number of nodes in the aggregation tree collecting data relevant to the query. For the AVG function we can compute the SUM aggregate and divide the resulting aggregate and the error guarantees by the value of the COUNT aggregate. Thus, for the computation of the AVG aggregate any changes to the COUNT aggregate due to node failures or nodes joining/leaving the network need to be transmitted at each epoch towards the monitoring node. As the work in [27] demonstrated, adaptive filter adjustment algorithms for the MAX and MIN aggregate functions make sense only when considering a multi-query optimization scenario. In single-query scenarios the optimal node setup for these two aggregates is always to allow a maximum deviation at the aggregate value computed at each node equal to the overall error guarantee. In subsect. 5.3 we discuss application of our algorithm to user-defined aggregate functions. The notation used in this section is summarized in Table 2.

In a *bandwidth-constrained* query evaluation, the user specifies the desirable *bandwidth utilization factor*

Table 2 Notation used in the description of our framework

Symbol	Description
B_Global	Average number of transmissions in the aggregation tree
B_Util	Input bandwidth utilization factor ($0 < B_Util < 1$)
E_Tot_i	Maximum deviation of the estimate at node N_i from the actual partial aggregate value
LTA_i	Last transmitted partial aggregate by node N_i to its parent
Err_i	Maximum deviation of the estimate at node N_i from LTA_i
W_i	Width or error filter at node N_i (W_i=2×Err_i)

$0 < B_Util < 1$. Similarly to the notation used in Sect. 3.2, let T denote the number of nodes in the aggregation tree. In an unconstrained evaluation, each node besides the *Monitor* node will transmit exactly one message (ignoring re-transmissions due to collisions), resulting in a total of $T-1$ transmissions per epoch. In a bandwidth-constrained evaluation, the system will process the query asserting that the average bandwidth consumption per epoch (that is the average number of transmissions over all nodes in the aggregation tree), denoted as *B_Global*, will be

$$B_Global = B_Util \times (T-1). \quad (1)$$

Since nodes do not transmit their measurements in every epoch, absolute accuracy is not feasible. Thus, our algorithm seeks to minimize the maximum possible deviation $|V-\hat{V}|$ of the aggregate value $\hat{V}$ that the *Monitor* node estimates as the result to the query from the actual current aggregate value V, when the overall *average* bandwidth consumption is *B_Global*. Furthermore, our algorithm will make this deviation available with each new estimate of the aggregate, providing strong error guarantees (ignoring message loss).

Description of our framework and node operation. At the initial phase, the *Monitor* node poses a query that is disseminated through the network in search of the sensor nodes that collect data relevant to the posed query. While each such node may have received the announcement of the query through multiple nodes, it only selects one of these nodes as its *parent* node, through which it will propagate its results towards the *Monitor* node. The flow of the query results forms a tree, rooted at the *Monitor* node, which is commonly known as the *aggregation tree* [14,19,24]. A sample aggregation tree is depicted in Fig. 2. The nodes in the aggregation tree can be classified as either *active* or *passive*. Active nodes (marked grey in the figure) collect measurements relevant to the query, while passive nodes (marked white in the figure) simply facilitate the propagation of results towards the *Monitor* node. An example of passive nodes arises when the posed query aggregates values observed by nodes in a limited, but distant from the *Monitor* node, area of the network. In this case the results will need to be propagated back to the *Monitor* node through nodes that will not collect data relevant to the query.

At each epoch, each sensor node N_i maintains an estimate of the partial aggregate of the measurements obtained by its descendant nodes in the aggregation tree. The node also keeps an error guarantee E_Tot_i that specifies the maximum deviation of this estimate from the actual partial aggregate value. The difference between

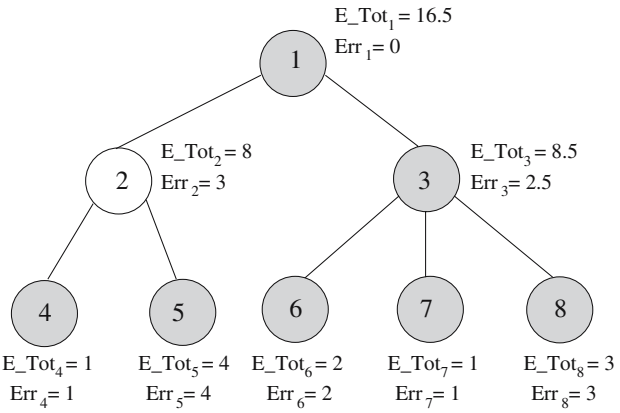


Fig. 2 Sample aggregation tree

the actual and the estimated partial aggregate values is due to the non-continuous transmission of data values from the sensor nodes due to the installation of *error filters* on them. An error filter is an interval of width $W_i{=}2{\times}Err_i$ that is centered around the last transmitted partial aggregate LTA_i by node N_i to its parent in the aggregation tree. At each epoch, node N_i transmit its estimated partial aggregate to its father, only if this value deviates by more than the filter's error Err_i from its last transmitted partial aggregate LTA_i (i.e., the new partial estimate lies outside its filter). We here need to emphasize that the values of E_Tot_i and Err_i are not the same. For example, for the SUM aggregate function,

$$E_Tot_i = \sum_{j:N_j \in \text{subtree}(N_i)} Err_j = Err_i + \sum_{j:N_j \in \text{children}(N_i)} E_Tot_j. \quad (2)$$

Example 2. Figure 2 shows the maximum error E_Tot_i and the error filters Err_i installed at the nodes, in our running example for the SUM aggregate function. For all leaf nodes in the tree, $E_Tot_i = Err_i$ since the deviation of the aggregate computed on a leaf node can only be as big as the node's filter. In this example, the measurement collected by node 5 cannot deviate by more than ± 4 from the last transmitted value to its parent node 2. Any larger deviation will result in a new transmission by node 5. In an intermediate node, the deviation of the true aggregate from the last transmitted value is due to all the error filters installed at the node's subtree. Thus, the maximum deviation of the aggregate in node's 2 subtree cannot exceed the sum of the filters in the nodes; i.e., $E_Tot_2 = Err_2 + Err_4 + Err_5 = 3 + 1 + 4 = 8$.

Through the proper adjustment of the filters' widths, our algorithm seeks to achieve an average overall bandwidth consumption of *B_Global* (a quantity derived at the initialization of the query from B_Util and the size

of the tree), while at the same time providing as tight error guarantees as possible. To accomplish the specified average bandwidth consumption, our algorithm has been designed to adjust the filter widths in cases when it detects that the bandwidth consumption is either above or below the given *B_Global* value, using a symmetric process that does not favor either the underutilization or the overutilization of the bandwidth. The decisions of which error filters to adjust are based on simple *local* statistics. Moreover, our algorithm is easy to configure since it contains a single parameter. Details on our algorithm for adjusting the filter widths are given in Sect. 5.

Example 3. Consider the subtree of the passive node 2 in Fig. 2, which is also depicted for clarity in Fig. 3. Assume that the last transmitted partial aggregate value of node 2 is $LTA_2 = 100$, and that at the first epoch of our example (left part of Fig. 3) nodes 4 and 5 transmit their new partial aggregate values (which are equal to the observations V_4 and V_5, correspondingly, of these nodes) $LTA_4 = 40$ and $LTA_5 = 59$. Given these values and the widths of the filters of these nodes as depicted in Fig. 2, the new intervals specified by the error filters in nodes 2, 4 and 5 now become:

- Node 2: $[100 - 3, 100 + 3] = [97, 103]$
- Node 4: $[40 - 1, 40 + 1] = [39, 41]$
- Node 5: $[59 - 4, 59 + 4] = [55, 63]$

The estimated partial aggregate value $\hat{V}_2$ at node 2 is equal to $99 = 40 + 59$ which lies within the interval $[97, 103]$ defined by the node's error filter and, therefore, the value 99 is not propagated towards the *Monitor* node.

Now consider that in the next epoch (right part of Fig. 3), the measurement of node 4 is 37 and the measurement of node 5 is 60. Node 4 will transmit the measurement 37 to node 2, and re-center its filter around that value: $[36, 38]$. Since node 5 will not make a transmission, the estimated partial aggregate $\hat{V}_2$ at node 2 is equal to $96 = 37 + 59$, which lies outside the node's error filter. Therefore, node 2 will transmit the value 96 to the *Monitor* node and set its filter to cover the interval $[93, 99]$.

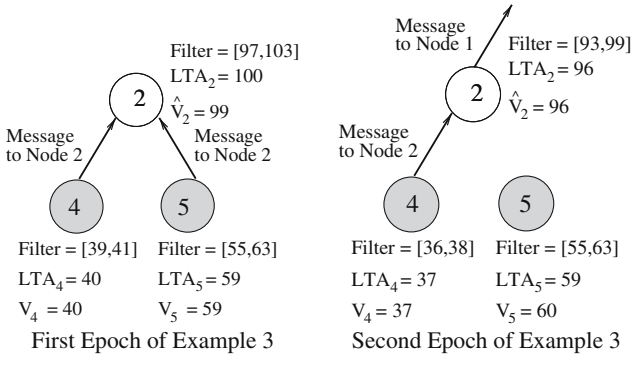


Fig. 3 Example of error filters

5 Our MGA algorithm

In this section we first describe the notion of a *marginal gain* and how it can be calculated at each node. We then provide details on which statistics need to be maintained at each node by our MGA algorithm, how each node calculates the cumulative bandwidth consumption within its subtree, and how our MGA algorithm dynamically adjusts the filter widths of the sensor nodes. Some basic information on the node operation and the use of the error filters was presented in Sect. 4. The notation used throughout this section is summarized in Table 3. A detailed description of these symbols is presented in appropriate parts of this section. We finally discuss some interesting extensions to our algorithm, including application to user-defined aggregate functions and modifications when node movement occurs or when local bandwidth constraints exist.

5.1 Algorithm description

Intuition of our algorithm. Assume a simple schedule in which the *Monitor* node decides how to adjust the nodes' error filters every Upd epochs. In our discussion herein we refer to each epoch in which the *Monitor* node decides to adjust the error filters as an *update epoch*. We will also use the term *update period* to denote the interval (of length equal to Upd epochs) between two

Table 3 Notation used in the MGA algorithm

Symbol	Description
B_Cum	Total bandwidth consumption in node's subtree
DE	Difference of filter widths between the two anchor points used when calculating the node's marginal gain
DB	Expected decrease in bandwidth when increasing filter width by *DE*
CumDE	Sum of DE values among all nodes with $DB \neq 0$ in subtree
CumDB	Sum of DB values among all nodes in subtree
budget	Total bandwidth (positive or negative) assigned to the node's subtree in the update process
Upd	Update period (number of epochs between adjusting the error filters)
B_{actual}	Overall bandwidth consumption since the last update epoch

consecutive update epochs, and the term *current update period* to denote the interval since the last update epoch.

Consider a snapshot of the network at an update epoch. Let B_{actual} denote the overall bandwidth consumption since the last update epoch. Let

$$B_Global \times \text{Upd} = B_Util \times T \times \text{Upd} \tag{3}$$

denote the targeted bandwidth consumption. If the network bandwidth is underutilized ($B_Global \times \text{Upd} > B_{\text{actual}}$), the *Monitor* node may instruct some nodes to increase their bandwidth consumption by decreasing their error thresholds. This necessitates the existence of a method to translate the additional bandwidth units in each node to changes in the node's error filter. An important question that is also being raised is which nodes should receive the most additional bandwidth? Since we want to provide tight error guarantees, it is evident that nodes which are expected to exhibit the largest reduction in their error filters (per additional bandwidth unit) should be ordered to increase their bandwidth consumption. Inversely, if the bandwidth is overutilized, then some nodes will be ordered to decrease their bandwidth utilization. In this case, the nodes which will exhibit the smallest increase in their error filters per reduced bandwidth unit should be ordered to have the largest decrease in their bandwidth consumption.

Marginal gains. In Fig. 4 we depict the expected width of a sensor node's error filter as we vary the desired number of transmitted messages by the node to its parent in the aggregation tree, within a period of Upd epochs. The maximum number of transmitted messages within an update period is obviously equal to Upd. This may occur, for example, when the filter has zero width and the partial aggregate value calculated by the node changes at each epoch. As the desired bandwidth consumption increases, the width of the filter that is expected to result in this bandwidth consumption gradually decreases.

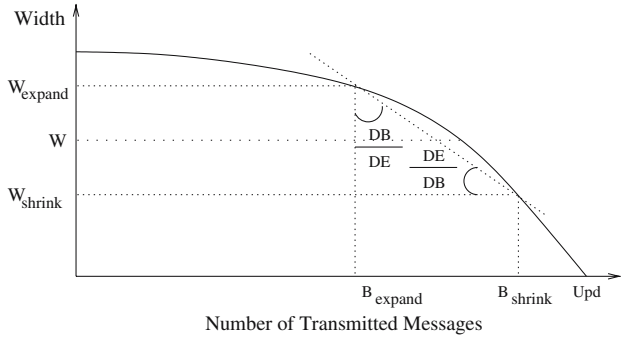


Fig. 4 Marginal gains of a node

Now, consider a randomly chosen node N_i in the aggregation tree and let W_i denote the node's error filter width. At each epoch, N_i decides whether to make a transmission, based on the value of the current partial aggregate calculated at N_i and its deviation from the previously transmitted aggregate value LTA_i (see Example 3.). At the same time, the node also keeps track of the number of transmissions it would have performed had its filter width been either a smaller $W_{\text{shrink}} < W_i$ or a larger $W_{\text{expand}} > W_i$ value. We refer to these filter width values as the two *anchor* points, and defer the discussion on how to determine their values for Sect. 5.2.

Let B_{shrink} and B_{expand} denote the calculated number of transmissions in each case, correspondingly, and also let

$$DB = B_{\text{shrink}} - B_{\text{expand}}, \tag{4}$$

$$DE = W_{\text{expand}} - W_{\text{shrink}}. \tag{5}$$

Then, the ratio $\frac{DE}{DB}$ is an indication of the decrease (increase) on the node's maximum error per additional (reduced) bandwidth unit assigned to the node.

Statistics maintained at each node. Besides the values of DB and DE that we described above, each node also needs to maintain some additional statistics. We denote as $CumDE_i$ the sum of the DE values among all nodes in N_i's subtree that have nonzero DB values; that is:

$$CumDE_i = \sum_{\substack{j\,:\,N_j \in \text{subtree}(N_i) \\ \text{and } DB_j > 0}} DE_j \tag{6}$$

$$= \begin{cases} DE_i + \sum\limits_{j\,:\,N_j \in \text{children}(N_i)} CumDE_j & DB_i > 0 \\ \sum\limits_{j\,:\,N_j \in \text{children}(N_i)} CumDE_j & DB_i = 0 \end{cases}$$

The reason why we exclude from the calculation of the $CumDE$ values those nodes which have zero DB values will be made clear later in this section. We also denote as $CumDB_i$ the corresponding sum of the DB quantities among all nodes in N_i's subtree. Each node can, therefore, perform the calculation of the $CumDB_i$ and $CumDE_i$ values using local statistics that its children nodes can piggyback to messages transmitted by them. This is a minimal amount of information needed that is aggregated at each node. These two quantities will be used, as we will explain later, by our algorithm to dynamically adjust the widths of the error filters. We here note that each node should not discard the latest individual $CumDB$ and $CumDE$ statistics transmitted by its children, as these quantities will also be used by our MGA algorithm in the allocation of bandwidth among the nodes in the aggregation tree.

Computing the bandwidth consumption. Because of the hierarchical topology and the limited transmission ranges of nodes, the *Monitor* node has no way of determining by itself the actual bandwidth consumption in the entire network. To resolve this, we use a simple intuitive idea. Each node maintains an estimate *B_Cum* of the overall bandwidth consumed by nodes in its subtree (including the node itself) during the current update period. When a node transmits a message to its parent node, it increments its calculated *B_Cum* value for its subtree by one (to account for the new message) and piggybacks this estimate in its message.[1] The parent in turn uses the last received bandwidth estimates from its children to calculate its own *B_Cum* value. Think of these values as "bubbles" that ascend the hierarchy when nodes transmit. The *Monitor* node sums-up all the values it receives from its children.

A small complication arises because some nodes in the middle of the hierarchy, due to their error filters, may have pruned messages. Thus, some statistics on the bandwidth consumption of their descendant nodes may not have been propagated towards the *Monitor* node. To solve this problem, at the epoch immediately before the invocation of the algorithm for adjusting the filters (discussed below), the statistics (bubbles) that still remain in the network are transmitted towards the *Monitor* node. Every node whose last transmitted value of *B_Cum* differs from the corresponding current value performs a transmission, even if this is not required by its latest measurement. This process goes on recursively until all bubbles reach the *Monitor* node. Note that the above procedure may only occur for non-leaf nodes of the aggregation tree.

Adjusting the error filters. We now present the complete MGA algorithm for dynamically adjusting the error filters installed in sensor nodes.

The algorithm starts at the *Monitor* node and progressively distributes additional bandwidth (which is positive in case of bandwidth underutilization, or negative in case of bandwidth overutilization) to subtrees and nodes in a top-down fashion. Each node N_i awaits a message containing the additional bandwidth $budget_i$ (positive or negative) to be distributed to the nodes in its subtree. If such a message arrives and $budget_i > 0$, then this budget is distributed among the node itself and the node's subtrees proportionally to the expected per bandwidth unit benefit of each choice, which is in turn equal to $\frac{DE_i}{DB_i}$ for the node itself (if $DB_i{>}0$) and $\frac{CumDE_j}{CumDB_j}$ for each child subtree with $CumDB_j{>}0$. Subtrees (nodes) having $CumDB{=}0$ ($DB{=}0$) receive no bandwidth and no message is being sent to them. For the remaining subtrees, their budget is calculated as (assuming $DB{>}0$):

$$\text{budget}_j = \frac{\text{budget}_i \times \frac{CumDE_j}{CumDB_j}}{\frac{DE_i}{DB_i} + \sum_{\substack{N_k \in \text{children}(N_i) \\ and\ CumDB_k > 0}} \frac{CumDE_k}{CumDB_k}} \tag{7}$$

and a message is transmitted to them with this value. The formula for calculating the budget allocated for the filter of node N_i itself is similar, but instead uses the quantity $\frac{DE_i}{DB_i}$ on the nominator. The budget allocated to the node is then multiplied by $\frac{DE_i}{DB_i}$ to determine the appropriate increase in the error filter's width. If the budget given to some subtree is very small (for example, less than 1) then there is no real benefit in such a budget assignment, since the update message itself will outweigh any possible benefits of error filter adjustments in the subtree. These small bandwidth budgets can be redistributed to subtrees which are programmed to receive additional bandwidth.

We now consider the case when either some node does not receive any update message, or has $DB = 0$. If $DB = 0$, then the node can decrease its error filter's width to W_{shrink} without this having an impact on the expected bandwidth consumed by the node. If $DB > 0$ and no bandwidth is allocated to the node, then the node does not modify its error filter.

The case when the bandwidth is overutilized (and therefore the disseminated budget is negative) is almost symmetric to our above discussion. The nodes which are expected to exhibit the smallest increase in their error filters per reduced bandwidth unit should be ordered to have the largest decrease in their bandwidth consumption. The distribution of the negative budget is performed proportionally to the $\frac{DB}{DE}$ quantity in this case. For any node N_i, the equation of allocating bandwidth to its children subtrees therefore becomes:

$$\text{budget}_j = \frac{\text{budget}_i \times \frac{CumDB_j}{CumDE_j}}{\frac{DB_i}{DE_i} + \sum_{N_k \in \text{children}(N_i)} \frac{CumDB_k}{CumDE_k}}. \tag{8}$$

The case when $DB = 0$ is handled almost identically to the case of bandwidth underutilization (a small difference is discussed in Sect. 5.2). Therefore, independently of whether the dispensed budget is positive or negative, the behavior of nodes having $DB = 0$ remains the same. This is the reason why they are not taken into account when calculating the *CumDE* values at each node. It is interesting to note that in the case of bandwidth

[1] Actually, as we explain in Sect. 6, it is often optimal, in terms of energy consumption, to transmit the *B_Cum*, *CumDE* and *CumDB* statistics only at the last epoch of each update period.

overutilization, the width of a node's error filter may either increase (if negative budget is assigned to the node), remain intact (if it receives zero budget), or even decrease (if $DB = 0$).

In either case (positive or negative bandwidth budget), each node N_i in the aggregation tree decides how to distribute its budget to the node itself and to its children subtrees by using only statistics received by its children nodes. The tree topology is taken into account when calculating the cumulative statistics (*CumDE*, *CumDB* and *B_Cum*) of each subtree. Finally, we need to note that in the first epoch after the reorganization, each node needs to transmit the new error E_Tot_i of its entire subtree (calculated bottom-up) so that the *Monitor* node will be able to know the error guarantees of its estimated aggregate value.

Example 4. We now present a simple example to demonstrate the error filter adjustment process. Consider the aggregation tree of Fig. 2 and assume that the budget of node 1 is 30 (the bandwidth was therefore underutilized in our example in the previous update period) and that the W, DE, DB, $CumDE$ and $CumDB$ values of each node are the ones presented in Table 4. Note that in two cases where $DB = 0$, the $CumDE$ values (marked in bold) have omitted from their calculations the DE values of some nodes. At the beginning, node 1 disseminates its budget based on the values $\frac{CumDE_2}{CumDB_2}$ and $\frac{CumDE_3}{CumDB_3}$. Using Eq. 7, the node dispenses budget 18 and 12 to nodes 2 and 3, respectively. The budget left for node 1 is 0, because there is no point in using a non-zero filter at the *Monitor* node. Now, considering just the case of node 2, the node will dispense budget 13.5 to node 4, 0 budget to node 5 and keep the remaining budget (4.5) for itself. Node 5 has $DB_5 = 0$ and, therefore, decreases its error filter to W_{shrink_5}. Node 4 will decrease its error filter by $0.9 = 13.5 \times \frac{2}{30}$ and set its new width to 1.6. Similarly, node 2 will decrease its filter by $0.1 = 4.5 \times \frac{2}{90}$.

Table 4 Sample statistics

Node	W	DB	CumDB	DE	CumDE
1	0	0	260	0	8
2	2.5	90	120	2	4
3	1.25	0	180	1	**4**
4	2.5	30	30	2	2
5	5	0	0	4	**0**
6	2.5	60	60	2	2
7	1.25	50	50	1	1
8	1.25	70	70	1	1

5.2 Algorithm details

We now present some details of our algorithm that are either not covered by the above discussion, or have been omitted to this point for ease of presentation.

Selecting the anchor points. While there is no optimal way to select these two anchor points W_{shrink} and W_{expand}, there are some useful guidelines that help determine their values. A simple technique would be to select filters with widths smaller and larger than W_i by a factor of a ($0 < a < 1$), i.e. setting $W_{\text{shrink}} = (1-a)W_i$ and $W_{\text{expand}} = (1+a)W_i$. However, using this setting, for nodes with small error filters the distance of the two anchor points will be small. This prevents the collection of useful statistics and often results in estimated DB values of zero. A more robust approach would utilize the standard deviation σ (or equivalently the variance σ^2) of the measurements collected at the node during the *previous* update period (the update period defined by the last two update epochs)[2] thus setting $W_{\text{shrink}} = \max\{W_i - \sigma, 0\}$ and $W_{\text{expand}} = W_i + \sigma$. This technique, however, is slow to react to nodes that exhibited a small variance in their measurements during their previous update period and which suddenly become more volatile in their measurements. What is, thus, needed is a combination of the aforementioned techniques.

In our MGA algorithm we use the following values for the two anchor points:

$$W_{\text{shrink}} = \max\{0, \min\{W_i - \sigma, (1-a)W_i\}\}, \quad (9)$$

$$W_{\text{expand}} = \max\{W_i + \sigma, (1+a)W_i, W_{\text{shrink}} + \min DE\}. \quad (10)$$

The $\min DE$ value specifies a minimum distance of the two anchor points and is needed in the case of nodes with both small filters and small variance in their measurements. Consider for example the case where the readings in a sensor node are integer values. Then, if the node's filter is centered around an integer value, any distance of the two anchor points smaller than 2 will always result in $DB = 0$. Thus, the $\min DE$ value can be determined by the granularity of the node observations.

Filter modification based on assigned budget. Let

$$DE_{\text{shrink}} = W - W_{\text{shrink}}, \quad (11)$$

$$DE_{\text{expand}} = W_{\text{expand}} - W. \quad (12)$$

Also, let DB_{shrink} (DB_{expand}) denote the expected increase (decrease) in the number of messages transmitted by the node by decreasing (increasing) its error

[2] The standard deviation at the current update period cannot be used, since this would result in moving anchor points, based on the observations during each, current, epoch.

filter to W_{shrink} (W_{expand}). Obviously, $DB = DB_{\text{shrink}} + DB_{\text{expand}}$. After determining the budget (positive or negative) assigned to each node during the update process, the desired modification of the error filter is more accurately calculated if, instead of using the node's DE and DB values calculated by both anchor points, the node utilizes just the statistics of the anchor point in the direction of the filter modification. Therefore, when zero budget is assigned to a node, the MGA algorithm shrinks the node's filter to W_{shrink} if $DB_{\text{shrink}} = 0$. If negative bandwidth is allocated to the node, then the ratio $\frac{DE_{\text{expand}}}{DB_{\text{expand}}}$ is used to determine the increase in the filter's width. If positive bandwidth is allocated to the node and $DB_{\text{shrink}}>0$, then the ratio $\frac{DE_{\text{shrink}}}{DB_{\text{shrink}}}$ is used to determine the decrease in the filter's width. Finally, if positive bandwidth is allocated to the node and $DB_{\text{shrink}} = 0$ (but $DB>0$), then the MGA algorithm shrinks the node's filter to

$$\min\left\{W_{\text{shrink}}, \max\left\{W - \frac{|budget| \times DE}{DB}, 0\right\}\right\}. \qquad (13)$$

We here need to emphasize that the DE_{shrink}, DE_{expand}, DB_{shrink} and DB_{expand} statistics are not transmitted to other nodes (to limit the size of the transmitted information), and that the bandwidth dissemination is based solely on the DE and DB values, and the corresponding cumulative statistics for the subtree. Furthermore, a straightforward extension would be to use more anchor points, which would also not be transmitted in order to limit the bandwidth consumption, but would solely be used to better estimate the modification of each node's error filter, based on the assigned budget to the node. However, such a modification would incur a computational cost proportional to the number of additional breakpoints. Even though this cost is usually small, we did not find such an extension to significantly impact the accuracy of the reported aggregate.

Additional details. The MGA algorithm also imposes a set of restrictions concerning the budget dissemination process.

- There is no point in assigning to a subtree negative budget larger (in absolute value) than the bandwidth *B_Cum* it consumed during the previous update period. Moreover, we cannot assign to a node itself more budget than what is necessary to drop its error to 0.
- For passive nodes with a single child node, the error filter is always set to zero, since one can easily demonstrate that it is always more beneficial to "push" the error budget of that node to its child.

5.3 Application to other aggregates

A question that naturally arises is whether the MGA algorithm can also be applied to a broader range of aggregate functions than the one described in Sect. 4. A nice classification of aggregate functions is presented in [24]. From the entire range of aggregate functions discussed in that paper, it would at first seem that the MGA algorithm could be applied to many *distributive* and *algebraic* aggregates. In short, in a distributive aggregate, the state that each node needs to maintain is the partial aggregate for the measurements in its subtree. An example of a distributive aggregate is the SUM function. On the other hand, in algebraic aggregates, the state that each node needs to maintain is not the partial aggregate for its subtree, but is of constant size. An example of an algebraic aggregate is the AVG function, where each node needs to maintain the SUM and COUNT aggregatess for its subtree.

The application of our MGA algorithm is not always straightforward (or even possible) in all such aggregates. In order for our algorithm to be effective, what is required is a way to calculate the cumulative statistic *CumDE* and the maximum error *E_Tot* at a node based solely on the corresponding values of its children nodes, and the DE and *Err* values, correspondingly, of the node itself. However, depending on the aggregate function, this is not always feasible.

Consider, for example, the aggregation tree of Fig. 5. In this tiny aggregation tree, the monitoring node simply computes the product of the values observed in nodes 2 and 3. If the reported values from these nodes have a maximum deviation of Err_2 and Err_3, respectively, then the estimated result in node 1 is $\hat{V}_2 \times \hat{V}_3$. Based on the *Err* values, and assuming that the reported aggregates are always non-negative, the minimum possible value of the true aggregate is $(\hat{V}_2 - Err_2) \times (\hat{V}_3 - Err_3)$, a value that can deviate at most by $Err_2 \times \hat{V}_3 + Err_3 \times \hat{V}_2 - Err_2 \times Err_3$ from the estimated aggregate. We thus have the complicated scenario where the maximum error depends on the values (aggregates) reported by the nodes in the aggregation tree, and not simply by the maximum errors of these nodes. Note that this behavior occurs at any subtree of the aggregation tree, and not solely at the *Monitor* node.

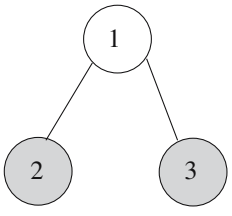


Fig. 5 Sample tiny aggregation tree

This, in turn, implies that to avoid violating the guarantees on the reported aggregate at the *Monitor* node, messages that may increase the error at a subtree may not be safely pruned, but may need to be propagated further up in the aggregation tree. This would, of course, significantly limit the benefits of our MGA algorithm, not only because fewer messages can be spared, but also because the need to transmit the maximum error of a subtree increases the size of each message.

An alternative solution would require the nodes to compensate, if possible, for changes in the maximum errors of their subtrees. If a node receives a message that indicates that the error guarantees at one (or more) of its subtrees have been modified, the node may consider whether it is possible to modify its error filter so that the guarantees of its entire subtree remain unaffected. This approach has the advantage that messages can still be effectively pruned even at lower levels of the aggregation tree. On the other hand, this approach would result in moving anchor points at each node and, therefore, significantly less reliable statistics.

Below, we present a family of aggregate functions that can be computed by our MGA algorithm. What is important, is that the issues described above may not always impact the operation of our algorithm if the error function is properly selected. In the example that we presented above, a proper choice would have been to represent the reported maximum error as a fraction of the estimated aggregate (instead of using a fixed absolute value).

5.3.1 User-defined aggregates

In what follows we formally define the requirements for running the MGA algorithm on a user-defined distributive aggregate function $Aggr(V_1, \ldots, V_T)$ calculated over the measurements collected by the T nodes in the aggregation tree.

Since *Aggr* is distributive, it must be easy to compute (i.e., with limited partial state required at each node) independently of the tree topology. A subtle result of this requirement is that the aggregate function *Aggr* cannot apply operators ('+', '−', '×', '/') of different precedence to *individual functions* of each measurement V_i. Examples of such individual functions are weighted functions (i.e., $a \times V_i$), exponential functions (i.e., V_i^a or a^{V_i}), logarithmic functions (i.e., $\log V_i$) etc.

To illustrate why *Aggr* may not simultaneously contain two operators of different precedence, consider the sample aggregation tree of Fig. 6 and let the desired aggregate function be: $Aggr = V_2 \times V_3 + V_4 \times V_6 + V_5 \times V_7$. In this case, no in-network aggregation can be performed, since the multiplications between pairs of measurements can be performed only when all these measurements have reached the *Monitor* node. Thus, the amount of state needed at each node when operators of different precedence exist is not bounded (as required by distributive functions), but rather depends on the exact definition of the aggregate function and the location of the nodes within the aggregation tree. Even if there exists an aggregation tree where such a function can be computed with a bounded amount of state at each node, changes to the aggregation tree (i.e., due to node movement or nodes joining/leaving the network) could violate the above condition.

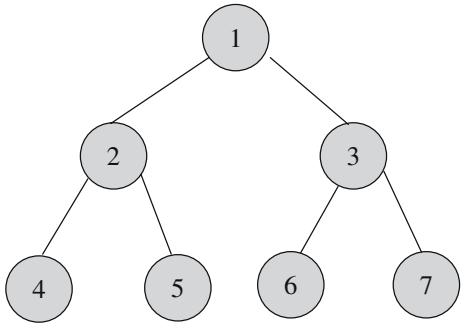


Fig. 6 Sample binary aggregation tree

Based on the above discussion, the aggregate function *Aggr* can either simultaneously contain: (i) the '+' and '−' operators; or (ii) the '×' and '/' operators. We discuss these two cases below.

Aggr contains the '+' and '−' operators. The first type of aggregate function *Aggr* is of the form

$$Aggr = \sum_i d_i \times f_i(V_i), \quad d_i = \{-1, 0, 1\}.$$

This is a case similar to the one discussed in our paper (a simpler form can be derived if the '−' symbols are incorporated in the individual $f_i()$ functions, so that *Aggr* is expressed simply as the sum of quantities). The case when $d_i = 0$ corresponds to passive nodes within the aggregation tree. Since the aggregate function is the sum of individual functions $f_i()$ applied to the measurements collected by the nodes, the error filters at each node can similarly be applied to the partial aggregate that the node estimates for its subtree. Note that the overall maximum error is in this case equal to the sum of the maximum errors over all the nodes in the aggregation tree.

Aggr contains the '×' and '/' operators. The second type of aggregate function *Aggr* is of the form:

$$Aggr = \prod_i (1\ OP_i\ f_i(V_i)), \quad OP_i = \{'\times', '/'\}.$$

In this case we first need to perform a transformation so that *Aggr* is expressed as simply the product of individual functions. This can be easily achieved if we replace each individual function $f_i()$ which is accompanied by $OP_i =' /'$ to $f_i'(V_i) = (f_i(V_i))^{-1}$. Using this set of transformations, we can similarly install error filters in all nodes of the aggregation tree. However, the boundaries of these filters will not be equidistant from the last transmitted partial aggregate, as in the case discussed in our paper. On the contrary, the distance of the upper and lower bound of each error filter from the last transmitted partial aggregate are expressed as a factor $Err_i \geq 1$ from that value:

$$H_i = LTA_i \times Err_i,$$
$$L_i = LTA_i / Err_i. \tag{14}$$

Note that using the above filter setup, the maximum error factor at the *Monitor* node is equal to $\prod_i Err_i$.

A small complication arises in this case at the dissemination of bandwidth to each node. It is easy to see that an increase dE to the error filter of a node with a small error filter results in a larger increase of the overall error than a corresponding increase at another node with a larger error filter. In particular, by increasing the error factor of node N_i by dE, the corresponding modification of the overall error factor is equal to:

$$\mathrm{d}E \times \frac{E_Global}{Err_i} \tag{15}$$

Thus, the decisions on how to disseminate the bandwidth cannot solely depend on the $\frac{DE}{DB}$ ratio (or the corresponding ratio of the cumulative statistics), but also need to take into account the error factor of each node. The only change that is, thus, required is to base the decisions on the $\frac{\frac{DE_i}{Err_i}}{DB_i}$ ratio and, instead of computing the *CumDE* statistics as in Section 5, calculate the corresponding sum of the $\frac{DE}{Err}$ quantities over all nodes in each subtree with $DB > 0$.

5.4 Extensions

We now describe interesting extensions to our framework.

Node movement. In sensor networks the aggregation tree often changes during the lifetime of a continuous query. This may happen because of node and link failures or, for instance, when nodes are attached to moving objects. When the aggregation tree gets reorganized, each node that experiences changes in the set of its children nodes needs to:

1. Receive the partial aggregate and collected statistics from its new children nodes.
2. Calculate the new partial aggregate and statistics of its subtree by considering the newly acquired values received from its new children nodes and by subtracting the corresponding values of its children nodes that were removed.
3. Make a transmission depending on the value of the new calculated partial aggregate.

Nodes joining/leaving the network. The case of a node joining the network is easily handled by assigning an error filter of zero width to this node. Thus, the error guarantees reported to the *Monitor* node are not violated. Moreover, the new node and its parent in the aggregation tree need to follow the same steps mentioned above (for the case of node movement), since the set of their children nodes has been modified.

The case of a node leaving the network is also dealt similarly but introduces some minor complications. In particular, when a node leaves the network the overall error guarantee becomes more tight. However, unless some notification is sent by the node leaving the network about its time of departure, the modification of the error guarantee cannot be performed until the next update period. The error guarantees reported in the *Monitor* node will not be violated in this case either, since the actual error guarantees will be tighter than the reported ones for a small period of time. Similarly, the reported estimate of the overall bandwidth consumption may be slightly lower than the actual one. Note that the estimate will not include the number of messages transmitted by the node leaving the network since the last transmission of its *B_Cum* value. Obviously, no such deviation will occur if the departing node notifies its parent in the aggregation tree about its departure. In this case the node's parent is aware of the exact number of messages that it has received by each of its children and can, thus, accordingly compensate and adjust its *B_Cum* value.

Strict bandwidth constraints in local areas. While our MGA algorithm limits, on the average, the targeted overall bandwidth consumption, it can be adapted to provide a strict bandwidth guarantee in all or parts of the aggregation tree. This will be useful when parts of the network exhibit severe bandwidth or energy constraints. As mentioned in the previous section, each node

maintains an estimate of the bandwidth consumption *B_Cum* in its subtree. If this bandwidth consumption exceeds the given strict constraint for the area, then sufficient negative budget should be assigned to this node's subtree in the next update epoch, independently of the overall bandwidth consumption.

Let B_Lim_i denote the bandwidth constraint in the subtree of node N_i. If we do not wish to impose a bandwidth constraint at certain subtrees, then $B_Lim_i = \infty$. Each node N_i needs to maintain an additional statistic B_Need_i which denotes the minimum amount, in absolute value, of negative budget needed to be disseminated to descendant nodes of N_i. The value of B_Need_i is calculated as follows:

$$B_Need_i = \sum_{k:N_k \in \text{children}(N_i)} B_Need_k + \max\{0, B_Cum_i - \sum_{k:N_k \in \text{children}(N_i)} B_Need_k - B_Lim_i\} \quad (16)$$

The first summand in the equation above represents the budget needed by nodes in subtrees rooted at children nodes of N_i. The second summand represents the budget needed by node N_i itself and is more complicated. To properly calculate this second summand, we need to take into account not only the bandwidth consumption and the bandwidth constraint in the entire subtree of N_i, but also consider how much of the difference between these two values will be offset by limiting the bandwidth consumed by the subtrees of N_i. The MGA algorithm now requires the following modifications:

- The *Monitor* node adds (since *B_Need* represents needed negative budget, in absolute value) its computed *B_Need* value to the budget that it will disseminate to the sensor nodes. Therefore, the disseminated budget is decided only after the *Monitor* node takes into account the minimum negative bandwidth needed by areas of the aggregation tree that have exceeded their bandwidth limits.
- Each subtree with a *B_Need* value greater than zero automatically receives at least this amount of negative budget. We here note that this negative budget will not be assigned to the subtree by its parent node, but is acquired automatically. Additional negative budget may be assigned to this subtree if either the budget disseminated by the *Monitor* node is negative, or if an ancestor node of this subtree with a non-zero *B_Need* value has at least one child subtree with a zero *B_Need* value. In the latter case, this ancestor node will disseminate negative budget to its subtrees, equal (only) to the max quantity that it calculates from Equation 16 (since the minimum negative budget needed by this node's subtrees is automatically acquired by them). Positive budget is disseminated only to subtrees that have not exceeded their bandwidth limit.

The above technique can also be used in situations when the desired bandwidth constraint needs to be imposed on an area of the aggregation tree (i.e., in nodes near the *Monitor* node). This requires the following steps:

- A new B_Cum' statistic is computed over the bandwidth consumed solely by the nodes in the region of interest.
- The *Monitor* node computes the negative bandwidth B_Need' that needs to be assigned to the nodes in the area of interest.
- The *Monitor* node calculates the bandwidth budget that it will distribute to the remaining nodes after taking into account the B_Need' value.
- Each node whose subtree contains at least one node in the region of interest performs two budget dissemination processes: one for the negative budget B_Need' to all subtrees that contain nodes in the area of interest and one for the bandwidth budget received for the subtree. The latter budget is disseminated towards all subtrees with at least one node outside the region of interest.
- For the remaining nodes, the bandwidth dissemination process is not modified.

Energy-aware bandwidth allocation. An important issue is whether the MGA algorithm can be adjusted in order to take into account the remaining energy at each node. Even at the initial stage of the posed query, there might be nodes in the network that possess low energy levels. These nodes should be spared, if possible, any unnecessary work. To limit the power consumption of such nodes one, or more, of the following approaches can be taken:

- Reduce the number of messages received and transmitted by these nodes. This can be achieved by installing large filters on the nodes themselves and to their immediate children in the aggregation tree. This approach has the drawback that the resulting error guarantees may be so large that the query will end up being of little use.
- Place such nodes in lower levels of the aggregation tree. Note that this is already done by many approaches (i.e., see [17]) that form and adjust the

aggregation tree based on the available energy of the nodes. For example, any node placed as a leaf will not incur in our approach any listening cost to receive partial aggregates by other nodes. Moreover, the number of its transmissions will depend solely on the node itself, and not on the values observed by any descendant node.

- Have these nodes join the aggregation process only in certain periods. For the remaining time intervals, in which the nodes may preserve energy by operating on a low-duty cycle mode, we can utilize a model (i.e., see [13,21]) to approximate their measurements.

Bandwidth reorganization in cases of stability. While the MGA algorithm does not guarantee an optimal bandwidth allocation policy to the nodes in the aggregation tree, its periodic dissemination of (positive or negative) bandwidth aims at reaching a good node setup that achieves bandwidth consumptions close to the target constraint.

However, it is possible that at some point of the operation of our MGA algorithm this periodic dissemination process will not occur for a long time, simply because the filter assignment to the nodes of the aggregation tree may result in a bandwidth consumption that matches (or is very close to) the target bandwidth consumption. This will result in stable node setups (except from those nodes with zero DB values).

While the aforementioned stability may be a result of stable data distributions of the collected data and good node setups, it might be possible to take a further optimization step. In particular, any non-leaf node with at least two child subtrees, and which has not received any amount of bandwidth units for several update periods, may initiate a local reorganization of the bandwidth consumed in its subtree. This process would include the following steps:

- Find the subtree i (subtree j) with the largest (smallest) ratio $\frac{CumDE}{CumDB}$. Consider only subtrees with non-zero $CumDB$ values. If these two ratios have similar values, do nothing.
- Otherwise, instruct subtree i to increase its bandwidth consumption by a given percentage (i.e., 20%) of the overall bandwidth consumption B_Cum_j of subtree j. To keep the expected bandwidth consumption in the entire subtree unaffected, also instruct the subtree j to reduce its consumption by the same percentage.

The above local reorganization process can prevent the MGA algorithm from remaining in bad filter setups that happen to achieve the target bandwidth consumption. However, we need to emphasize that even with stable distributions of the monitored data the above situation is expected to happen very rarely. In all of our experiments (even with synthetic data sets with stable distributions over time) the overall bandwidth consumption continuously fluctuated around the target bandwidth constraint. Thus, the scenario where the filter adjustment process was not invoked for several update periods did not happen, even in this case.

6 Alternative techniques

We now provide a description of an alternative technique that can potentially be used, for the evaluation of bandwidth-constrained aggregate queries. The *threshold-based adjustment* (TBA) algorithm presented below is motivated by the work of Olston et al. [30] on determining when cached objects should be refreshed based on a defined priority function. A detailed description of the algorithm and justification of its decisions can be found in [30]. Compared to the algorithm described in [30] for non-hierarchical node settings, our modified version of the algorithm has two major differences:

- Nodes may be required to transmit their computed partial aggregate due to transmissions performed by their children nodes.
- The adjustments of the node filters can only be performed periodically, since in a hierarchical setting we need to get accurate estimates of the actual bandwidth consumption.

These issues are discussed below and in Sect. 7.1.

We need to note that we also experimented with some additional techniques that can also be used in our application. These techniques are based on either uniform or biased sampling of the data sources. However, due to the inability of these alternate techniques to provide strong deterministic error guarantees and their poor performance in our experimental evaluation, we omit their discussion.

6.1 Threshold-based adjustment (TBA)

We now describe how the TBA algorithm can be adapted for the problem of bandwidth-constrained queries over sensor networks. Each active node i in the aggregation tree maintains the following statistics at each epoch t_i:

- The value $V_{t_{\text{now}}}$ of the node's monitored quantity at the current epoch.

- The last transmitted measurement $V_{t_{\text{last}}}$ of this node and the time (epoch) of the last transmission t_{last}.
- A threshold value Thr_i that will help determine the time of the node's next transmission.
- A priority value Pr_i calculated as:

$$Pr_i = (t_{\text{now}} - t_{\text{last}}) \times \max_{t \in [1+t_{\text{last}}, t_{\text{now}}]} |V_t - V_{t_{\text{last}}}| - \int_{t_{\text{last}}}^{t_{\text{now}}} |V_t - V_{t_{last}}| \mathrm{d}t. \quad (17)$$

We here note that the definition of the priority function has been slightly modified (to include the *max* quantity) from the formula in [30], since in our case, the deviation value $|V_t - V_{t_{\text{last}}}|$ is **not** a non-decreasing function of t. The *max* quantity is therefore needed to ensure that the calculated priority is always an non-decreasing and non-negative value, as required in [30].

Each time a node's priority value Pr_i exceeds the node's threshold value Thr_i, the node performs the following operations:

- Increases its threshold value by a factor θ (i.e., θ=1.1 in our experiments); that is: $Thr_i = Thr_i \times \theta$. The parameter θ controls how quickly the node will slow down its transmission rate, in the absence of feedback received by the *Monitor* node.
- Transmits the difference $V_{t_{\text{now}}} - V_{t_{\text{last}}}$ and its current threshold towards its parent node in the aggregation tree.
- Sets $V_{t_{\text{last}}} = V_{t_{\text{now}}}$, $t_{\text{last}} = t_{\text{now}}$ and $Pr_i = 0$.

Because the threshold values are applied to the node's observed value, and not to its partial aggregate, in a hierarchical node setting each node N_i may also transmit the difference $V_{t_{\text{now}}} - V_{t_{\text{last}}}$ if one of its descendant nodes makes a transmission. In this case, node N_i can include the above difference at no cost (after aggregating it with the one received by its child node), since any messages received from descendant nodes will need to be propagated towards the *Monitor* node anyway. In this case, the node performs most of the steps described above, but does not increase its threshold value (neither transmits it), since its transmission was due to another node's measurements.

The *Monitor* node monitors the thresholds received from the nodes and, if it detects that the bandwidth is underutilized (the estimation of the bandwidth consumption can be performed in the same way as in our algorithm), then it sends feedback messages to the nodes with the highest thresholds to divide their thresholds by a parameter ω>1. This ω parameter controls how aggressively the *Monitor* node wants each notified node to increase its transmissions. While this adjustment process can be performed in a continuous fashion in [30], in hierarchical node settings this process can be performed only periodically, since after each adjustment phase we need to collect accurate estimates of the resulting bandwidth consumption. Trying to keep the thresholds of all active nodes about equal was shown in [30] to be the optimal solution, for a different problem though than the one that we tackle in this paper, and in a non-hierarchical setup of the nodes.

6.2 Comparison to TBA algorithm

We now provide a classification of the TBA algorithm described in Sect. 6.1 based on several characteristics and draw comparisons to the characteristics of our MGA algorithm. Table 5 provides a synopsis of the characteristics of these two algorithms.

Provided error-guarantees. The TBA algorithm provides deterministic error guarantees, since at each node N_i, its current value cannot deviate by more than Thr_i without resulting in a transmission by the node. Therefore, the application error guarantee is equal to the sum of threshold values by all active nodes in the aggregation tree. Our MGA algorithm also provides deterministic error guarantees, as discussed in previous sections, but is significantly more accurate, as our experiments will demonstrate.

Robustness to nodes with different characteristics. In a large sensor network, nodes with widely different characteristics may exist. For example, the measurements of some nodes may either be significantly higher or exhibit much larger variance than the measurements of some other nodes. The TBA algorithm is only influenced by the variance of the measurements, and not by their magnitude. However, the TBA algorithm fails to take into account that some nodes may be *erratic*, meaning nodes that exhibit large variance in their measurements. Such nodes tend to continuously make transmissions, since their thresholds are usually not sufficiently large to prune any messages. On the update phase of the TBA algorithm, due to their increased threshold values, these same nodes are the ones which the *Monitor* node will ask to lower their thresholds. Non-erratic nodes will

Table 5 Characteristics of TBA and MGA

Technique	Error guarantee	Robust	Utilizes hierarchy	Side info
TBA	YES	NO	NO	Considerable
MGA	YES	YES	YES	Limited

gradually increase their thresholds to large values, thus resulting in large maximum errors for the application. However, it is obvious that the desired behavior would be for the algorithm to eliminate the erratic nodes from consideration (by assigning them a zero, or near-zero threshold). This would still result in a continuous transmission by these nodes, but the thresholds of the remaining nodes would be considerably lower, returning tighter error guarantees.

Our MGA method is not influenced by the existence of sets of nodes with significantly different magnitude of values. Furthermore, it is robust in dealing with erratic nodes, which it easily manages to eliminate from consideration. Erratic nodes tend to have, due to their large variance, small DB values. Any erratic node with a large error filter will exhibit a large $\frac{DE}{DB}$ ratio. This makes it a good candidate for receiving a lot of positive bandwidth and, therefore, significantly decrease its error at the first epoch when bandwidth is underutilized. Moreover, in cases of bandwidth overutilization, little negative bandwidth, if any, is given to erratic nodes with wide error filters, due to their small $\frac{DB}{DE}$ ratio.

Hierarchy consideration. One of the significant drawbacks of using the TBA algorithm for evaluating bandwidth-constrained queries is the fact that it does not exploit or use the hierarchical structure of the aggregation tree. Deciding the set of nodes whose threshold values will be decreased is based solely on the threshold values, and not on the topology of the nodes. However, the tree topology should clearly be taken into account, since the transmission by any node N_i causes the transmission of messages by all the ancestors of N_i in the aggregation tree. On the other hand, our MGA algorithm takes into account the tree hierarchy in the way that it calculates the cumulative statistics of each subtree, and then uses this information in the update phase to dispense the available budget.

Amount of side information needed. We now consider the amount of additional information, besides the propagated aggregate values, that the presented techniques need in order to make their decisions.

In the TBA algorithm, each message from node N_i to its parent node is accompanied by a potentially large list of identifiers of nodes which increased their thresholds in the subtree of N_i. This information may be quite large, since it cannot be aggregated. Moreover, the algorithm needs to estimate the amount of consumed bandwidth, which requires some additional, but easy to aggregate, information to be propagated as well. Our MGA algorithm requires the transmission of the B_Cum, $CumDE$ and $CumDB$ quantities which, however, is information that can be easily aggregated.

We here need to emphasize that the statistics by both the MGA and TBA algorithms can be transmitted only periodically (i.e. in the last epoch of the update period) to limit the amount of transmitted information and, therefore, the energy drain in nodes. This is optimal when using the cost model used by the LEACH [17] and Pegasis [23] protocols, where the energy drain during the transmission or reception of messages is proportional to the number of transmitted/received bits. In this case, the TBA algorithm will also adjust the thresholds of nodes only periodically.

7 Experiments

We have developed a simulator for testing the TBA and MGA algorithms that we discussed in this paper under various conditions. The experiments that we present are split in two parts. We first use synthetic data sets to test the effect of various data characteristics. In the second part we experiment with real sensor data.

7.1 Configuration parameter selection

We first ran a set of preliminary experiments for setting up the configuration parameters of the algorithms. Both algorithms had a large range of values for their configuration parameters that provided near-optimal results. As an example, we show in Figs. 7 and 8 the average error guarantee when varying the update period (Upd) from 10 to 100 epochs for the light and temperature measurements of the lab data set discussed in Sect. 7.3. Using these results, we set the update period Upd for TBA to 50 and for MGA to 40 epochs.

For TBA we found out that the parameters that were originally proposed in [30] provided the best results in most cases. We thus used initial values of θ=1.1 and

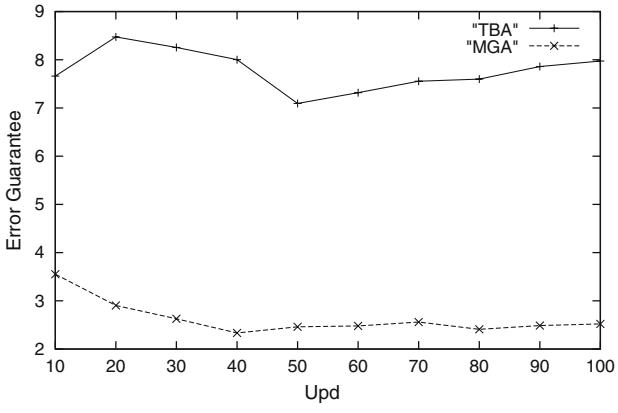


Fig. 7 Error Guarantee varying Upd, temperature measurements (lab data set)

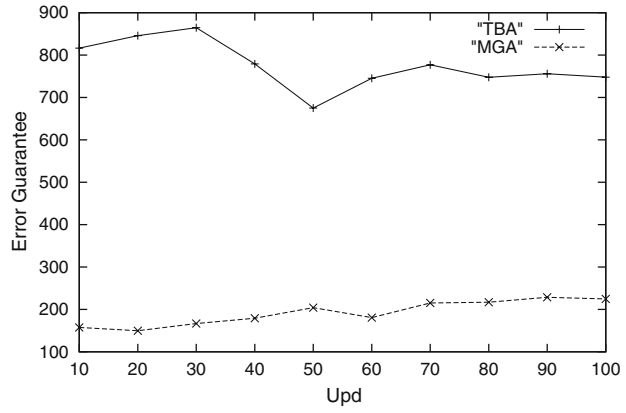


Fig. 8 Error Guarantee varying Upd, light measurements (lab data set)

Table 6 Error Guarantee, varying α (lab data set)

α	Temperature	Light	Humidity
0.01	2.49	225.71	4.64
0.05	2.44	213.11	4.40
0.1	2.55	188.15	4.28
0.2	2.57	192.93	4.20
0.3	2.33	179.04	4.31
0.4	2.39	189.69	4.08
0.5	2.35	183.64	4.35
0.6	2.31	214.39	4.27
0.7	2.52	237.44	4.21
0.8	2.51	301.47	4.39
0.9	2.33	391.68	4.98
0.95	3.54	513.78	6.86
0.99	5.70	604.51	8.77

ω=10. However, unlike the setting in [30], in hierarchical topologies the amount of bandwidth can be measured only periodically, unless statistics about the bandwidth consumption are propagated continuously (which would impose a significant bandwidth and energy overhead). This fact significantly complicates the identification of the correct parameters for the TBA algorithm in each case, as one would expect the ω value to exhibit a dependency on the period Upd between two consecutive update operations. In order to achieve the specified bandwidth constraint, we thus had to periodically adapt these values during the execution of the TBA algorithm based on the actual and the desired bandwidth consumption. In cases of bandwidth underutilization, the number of nodes that were chosen to lower their thresholds was increased by 11%, while this number was decreased by 12% in cases of bandwidth over-utilization. Finally, if the number of updated nodes exceeded half the number of total nodes in the tree, the value of ω was multiplied by a factor of 4. We found out that this setting was providing the best results for TBA.

The MGA algorithm is not very sensitive in the used value of the α parameter discussed in Sect. 5.2. As an example, in Table 6 we show the error guarantee of MGA for a lab data set, described analytically in Sect. 7.3, which consists of a trace of readings from sensors in the Intel Research, Berkeley lab [13], collecting light, humidity and temperature readings. We varied for this data set the α parameter from 1 to 99% when the value of *B_Global* was set to 6. Based on these results, we simply set α=40% in all our experiments. Our algorithm is not very sensitive in the value of the parameter, as long as this parameter is not set too low (values below 5%), or too high (80% and above). Note that this sensitivity is observed only for the light data, where the filters are relatively larger. For the temperature and humidity data sets the filters are so small that the anchor points are almost always determined by either the *minDE* parameter or the variance of the measurements. For the light data set, the lower sensitivity that we observe for smaller, when compared to larger, values of α is because for small values of α the anchor points are usually determined by the variance of the measurements.

In all cases we used the first 10% of the epochs as a warm-up period, in order for the algorithms to adjust the filters/thresholds of the nodes. This period was also enough for the TBA algorithm to discover a good ω value for each bandwidth constraint.

7.2 Sensitivity analysis

There are two orthogonal dimensions that affect the evaluation of a bandwidth constrained query. The first is the hierarchical organization of the nodes and the second is the data distribution. We have experimented with several topologies for the aggregation tree. For brevity we present results for the following configurations.

- $T1$: In this configuration the sensor nodes form a balanced tree with fanout = 3 and 6 levels (364 nodes overall). Only leaf nodes in the tree are active. Intermediate nodes do not collect measurements but rather aggregate results from their subtrees.
- $T2$: This is like $T1$ but now all nodes in the tree collect measurements.
- $T3$: The nodes form a random tree with the fanout of each node being randomly chosen between zero (leaves) and 8 and with the maximum distance of a leaf from the *Monitor* node equal to 6. The tree is not balanced and leaf nodes are in different distances from the *Monitor* node. The tree that we used had 644 nodes. Intermediate nodes are active with probability 20%. All leaf nodes are active by default.

Table 7 Avg Error Guarantee, Avg Relative Error Guarantee, Avg Abs Error and B_used for $T1$

	MGA				TBA			
		Error guarantee				Error guarantee		
B_Global	B_used	Absolute	Relative (%)	Abs. error	B_used	Absolute	Relative (%)	Abs. error
20	19.5	3,667.0	1.44	279.2	19.0	35,013.7	13.76	245.9
30	28.9	2,506.8	0.99	195.7	28.5	9,527.5	3.74	203.7
40	38.6	1,914.2	0.75	155.4	38.1	5,221.6	2.05	217.1
50	48.4	993.1	0.39	75.7	46.8	4,255.7	1.67	218.6
60	58.0	442.2	0.17	46.9	55.7	3,439.0	1.35	208.1
70	67.5	217.6	0.09	21.4	62.2	2,438.8	0.96	161.5
80	77.7	121.5	0.05	10.9	72.7	1,417.7	0.56	101.7
90	87.8	71.5	0.03	5.7	77.4	1,463.2	0.58	107.2
100	97.6	26.1	0.01	1.8	89.9	675.0	0.27	54.1

The synchronization of the sensor nodes in the tree is performed as described in TAG [24]. TAG reduces the number of messages by combining, whenever possible, messages on a path to the *Monitor* node within the same epoch. All algorithms are implemented on top of this protocol.

In the synthetic data sets, we had the values of each active node follow a random walk pattern. The maximum size of each step and the frequency of the steps are controlled as follows. For the maximum step size we split the nodes in two classes: *regular* nodes and *erratic* nodes. Regular nodes have a maximum step size of 2, while erratic nodes make much larger maximum steps, up to 200 each time. Regarding the frequency of the steps (that is an orthogonal aspect of the data) we had two additional classes: *sleepers* and *restless* nodes. Sleepers make very infrequent moves with a low probability of 1%. Restless nodes make a random step at every epoch. In our experiments we will vary the mix of regular/erratic and sleepers/restless nodes and study their effects on the algorithms.

For the first set of experiments we used the SUM aggregate function. In Sect. 7.3 we present results using the AVG function.

Effect of bandwidth constraint. In Table 7 we show the average error guarantee in absolute and relative terms (over the SUM aggregate) and the average absolute error provided by the algorithms over 10,000 epochs for different values of *B_Global* (bandwidth constraint) and for the configuration T1. The corresponding values of B_Util can be easily derived in each case by considering the number of nodes in each tested configuration. The data we used had a 80/20 mix of regular/erratic nodes (that is 80% of nodes were regular and 20% erratic) and, similarly, a mix of 80/20 of sleepers/restless nodes.

In this table, the first column shows the constraint used, while column *B_used* shows the average bandwidth (number of messages per epoch) achieved per algorithm.[3] The numbers also include any control messages required by the algorithms. We notice that both algorithms achieve a bandwidth consumption very close to the input value. However, there are large differences in the error guarantee provided per algorithm. The error guarantee of algorithm MGA is up to 26 times smaller than the one of TBA for the same bandwidth constraint. In this table we also show the average absolute deviation of the reported aggregate to the *Monitor* node from the true aggregate value (corresponding to an unconstrained execution). We notice that both algorithms are significantly more accurate than their reported error guarantee; the (real) absolute error is typically an order of magnitude smaller than the reported error guarantee. This is a trend consistent in all our experiments. For brevity, in the remainder of this section we are only reporting the error guarantees. Also, note that, as expected, with

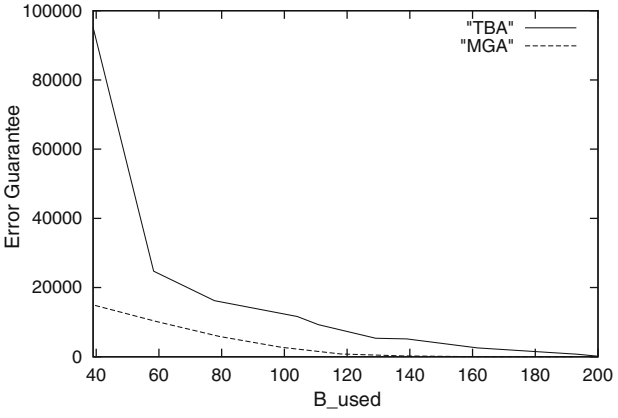


Fig. 9 Error guarantee for $T3$

[3] As mentioned, *TBA* does not aggregate the statistics sent to the *Monitor* node and, thus, the size of each message is substantially larger. We do not account for this overhead of *TBA* here.

Table 8 Varying mix of regular/erratic nodes

Regular/erratic	$T1$ (%)		$T2$ (%)		$T3$ (%)	
	MGA	TBA	MGA	TBA	MGA	TBA
0.1	0.12	0.80	0.31	1.09	0.91	2.22
0.2	0.26	1.69	0.78	2.07	1.95	4.39
0.3	0.76	2.65	1.50	2.69	2.52	7.02
0.4	1.31	3.83	2.59	3.49	2.74	10.39
0.5	1.72	4.55	2.94	3.78	3.33	15.97
0.6	1.88	4.33	4.21	6.12	4.26	24.12
0.7	3.30	6.92	4.29	7.33	4.75	30.88
0.8	3.63	9.48	5.06	15.01	5.42	37.66
0.9	3.96	9.49	5.44	16.71	6.06	43.93

Table 9 Varying mix of sleeper/restless nodes

Sleeper/restless	$T1$ (%)		$T2$ (%)		$T3$ (%)	
	MGA	TBA	MGA	TBA	MGA	TBA
0.1	0.02	0.22	0.09	0.41	0.27	1.17
0.2	0.26	1.69	0.78	2.07	1.95	4.39
0.3	1.21	3.40	2.31	3.19	2.98	8.92
0.4	1.77	4.23	4.76	5.46	4.27	16.51
0.5	3.28	7.02	7.08	8.11	6.09	27.61
0.6	4.93	10.56	8.14	11.25	7.75	42.29
0.7	6.53	14.19	9.80	17.41	10.04	54.20
0.8	9.19	23.21	12.66	29.80	12.07	71.34
0.9	11.93	30.26	13.15	34.89	15.45	98.83

the increase of the desired bandwidth consumption the error guarantees become tighter. Some deviations from this behavior may be observed rarely in cases when the error is very low, since the average error guarantees depend on the actual bandwidth utilization during each update period.

In Fig. 9 we plot the average error guarantee versus *B_Global* for $T3$. The error guarantee of MGA, for the same bandwidth constraint is smaller by a factor of up to 75, depending on the used bandwidth constraint.

Table 10 Average relative error guarantee, varying P_{move}

P_{move}	$T1$ (%)	$T2$ (%)	$T3$ (%)
0	1.06	2.63	2.38
0.1	1.19	2.75	2.71
0.2	1.27	2.87	2.73
0.3	1.34	2.92	3.39
0.4	1.45	2.80	2.91
0.5	1.75	3.01	3.53
0.6	2.14	3.06	3.49
0.7	1.94	3.62	3.88
0.8	2.16	3.75	3.44
0.9	1.99	3.21	4.16
1.0	2.55	4.45	5.21

Overall, the error guarantees provided by the algorithms are very tight. The average value of the SUM aggregate was 254,429 and 531,718 for $T1$ and $T3$ ($T1$ has fewer active nodes than $T3$). For *B_Global*=20 in $T1$, MGA provides an average error guarantee of 3,667. In relative terms this is just 1.5% of the aggregate value and is obtained with just 19.5 messages per epoch, while an unconstrained execution of the query, as in [24], would require 363 messages per epoch. Thus, we obtain a 1.5% error guarantee using about 5% of the bandwidth. The actual error is just 0.1% for the same bandwidth.

Varying mix of nodes. In Table 8 we vary the proportion of regular over erratic nodes and present the error guarantee as a percentage of the real aggregate. The proportion of sleepers/restless nodes was 80/20 and *B_Global*= 60 in all cases. As the ratio of regular/erratic nodes decreases, the error guarantees increase because of the increased number of erratic nodes. Clearly, the error guarantee provided by MGA are significantly smaller than the ones provided by TBA.

In Table 9 we repeat the experiment varying this time the mix of sleepers/restless nodes. The ratio of regular/erratic nodes was 80/20 and *B_Global*=60.

Node movement. In Table 10 we experiment with the sensitivity of the MGA algorithm when the aggregation

Table 11 Characteristics of real data sets

Measure	Mean	Variance (data measurements)	Variance (AVG aggregate)
Lab-temperature (Celsius)	21.81	11.55	1.72
Lab-light (Lux)	378.61	254,668.00	11,718.55
Lab-humidity (0-100%)	38.53	40.96	5.63
Weather-temperature (Fahrenheit)	70.25	225.93	2.54

tree changes during the execution of the query (the ratios of regular/erratic and sleepers/restless nodes were both set to 80/20, while B_Global=40).

We model node movement as follows. Between two update periods, with probability P_{move} a node N_i in the tree changes its selection of parent node, by randomly choosing a new parent from the tree (excluding of course its descendants). This results in having the whole subtree rooted at N_i moved to a new location. The main implication of this change is that during the next update, statistics accumulated by ancestors of N_i in the old/new configuration do not accurately depict the behavior of the nodes before and after the change in the topology. However, from the table we can see that the effect on the accuracy of MGA is not very large, even when the aggregation tree changes rather frequently. We also observe that the reported errors are not strictly monotonic with P_{move} as they depend on both the selection of N_i and its randomly selected new location on the tree.

7.3 Experiments with real data

Sensor networks are frequently used in environmental monitoring. Here, we present results on running experiments over two real-world data sets. The first data set, *lab*, is a trace of readings from sensors in the Intel Research, Berkeley lab, collecting light, humidity and temperature readings [13].[4] We used trace data of 48 sensors for a period of 1 week. For setting up the aggregation tree, we used the aggregate connectivity data available with this trace. The sensors formed a tree through a simple protocol that prioritized the choice of a parent node based on the quality of the upload link. The final aggregation tree is shown in Fig. 10.

The second data set, *weather*, provides temperature measurements at a resolution of one minute for the year 2002 (a total of 525K measurements). The data was collected at the weather station of the university of Washington as part of the Live From Earth and Mars Project. We split the data into 53 non-overlapping sets. We tested this set using a random aggregate tree consisting of 80 nodes. Each intermediate node in the tree had between one and four children while the maximum distance of a leaf from the *Monitor* node was 5 (i.e., 6 levels). Intermediate nodes were active with probability 20%.

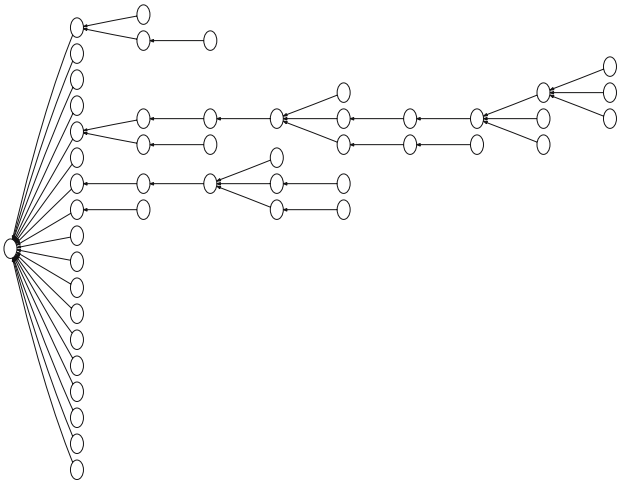


Fig. 10 Aggregation tree used in *lab* data set

For the interpretation of the performance of the algorithms we provide in Table 11 the mean value and variance of the sensor measurements (calculated over all the nodes) as well as the variance of the AVG function (calculated at the *Monitor* node) used in all the aggregate queries in this subsection. In Figures 11, 12 and 13 we

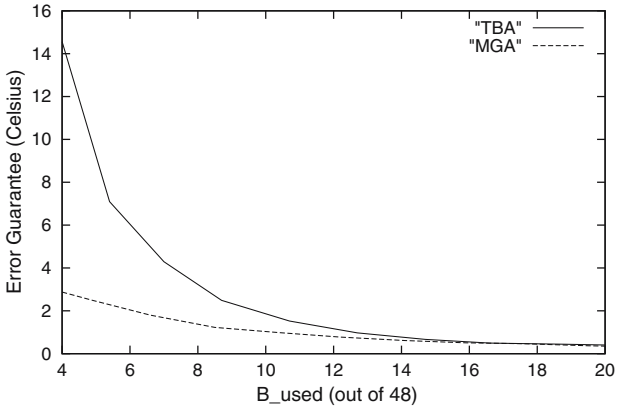


Fig. 11 Error guarantee for temperature readings (lab data set)

[4] We would like to thank the authors for making their data publicly available.

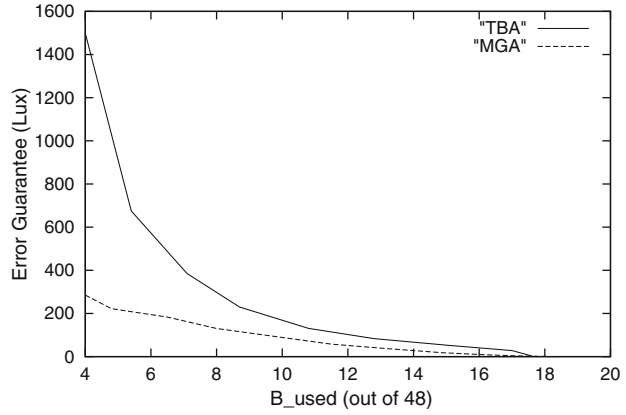


Fig. 12 Error guarantee for light readings (lab data set)

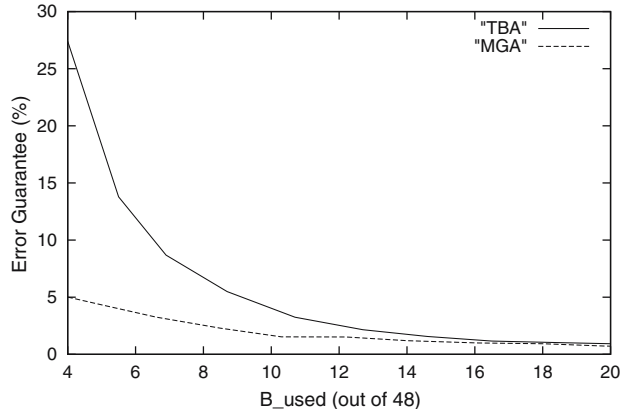


Fig. 13 Error guarantee for humidity readings (lab data set)

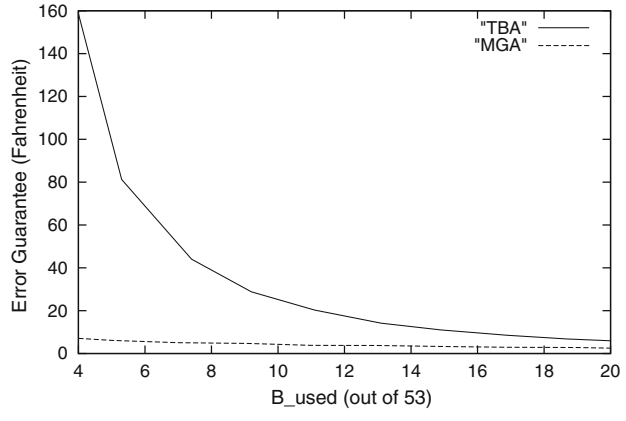


Fig. 14 Error guarantee for *weather* data set

show the average error guarantee for the lab data set, while Fig. 14 depicts the performance for the weather data set. We notice that with about 10% of the readings, MGA provides strong deterministic guarantees that are below the variance of the aggregate in most cases.

8 Conclusions

In this paper we introduced the notion of bandwidth constrained queries in sensor networks. We proposed a new algorithm that seeks to minimize the maximum error of the approximation, while keeping the average bandwidth consumption under a given threshold. We also discussed extensions that allow us to meet local bandwidth constraints in the network. Unlike prior work, our techniques provide strong deterministic guarantees for the deviation of each newly estimated aggregate and make these guarantees available to the monitoring node at each epoch. The algorithm operates with simple but intuitive local statistics that are maintained by the nodes. Dissemination of positive or negative bandwidth is done in a localized manner between parent and child nodes in the aggregation tree. While our experiments have demonstrated that our algorithm is substantially more accurate than alternative techniques (often by more than an order of magnitude) the strongest result is that quite often we can compute a very accurate approximation of the requested aggregate with just a fraction of the bandwidth that an unconstrained computation of the aggregate would require. Thus, bandwidth constrained queries can have a dramatic effect in (i) reducing the bandwidth consumption of continuous queries, allowing the network to handle a lot more queries for the available bandwidth and (ii) in case nodes are operated by batteries, prolonging the longevity of the network by reducing the number of messages exchanged among the nodes of the network.

References

1. Barbará, D., Garcia-Molina, H.: The Demarcation Protocol: A Technique for Maintaining Linear Arithmetic Constraints in Distributed Database Systems. In: EDBT, 1992
2. Bawa, M., Garcia-Molina, H., Gionis, A., Motwani, R.: Estimating Aggregates on a Peer-to-Peer Network. Technical report, Stanford, 2003
3. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In: INFOCOM, 2002
4. Chang, J-H., Tassiulas, L.: Energy Conserving Routing in Wireless Ad-hoc Networks. In: INFOCOM, 2000
5. Chen, J., Dewitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In: ACM SIGMOD, 2000
6. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over Imprecise Data. In: SIGMOD, 2003
7. Cheng, R., Prabhakar, S.: Managing uncertainty in sensor databases. SIGMOD Rec., **32**(4):41–46, (2003)
8. Considine, J., Li, F., Kollios, G., Byers, J.: Approximate Aggregation Techniques for Sensor Databases. In: ICDE, 2004
9. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Compressing Historical Information in Sensor Networks. In: ACM SIGMOD, 2004

10. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Hierarchical In-Network Data Aggregation with Quality Guarantees. In: EDBT, 2004
11. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Dissemination of Compressed Historical Information in Sensor Networks. VLDB J., (2007). DOI 10.1007/s00778-005-0173-5
12. Demers, A., Gehrke, J., Rajaraman, R., Trigoni, N., Yao, Y.: The cougar project: a work in progress report. SIGMOD Rec., **32**(4):53–59 (2003)
13. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M., Hong, W.: Model-Driven Data Acquisition in Sensor Networks. In: VLDB, 2004
14. Estrin, D., Govindan, R., Heidermann, J., Kumar, S.: Next Century Challenges: Scalable Coordination in Sensor Networks. In: MobiCOM, 1999
15. Ghose, A., Grossklags, J., Chuang, J.: Resilient Data-Centric Storage in Wireless Ad-Hoc Sensor Networks. In: Mobile Data Management, 2003
16. Heidermann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., Ganesan, D.: Building Efficient Wireless Sensor Networks with Low-Level Naming. In: SOSP, 2001
17. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In: HICSS, 2000
18. Hellerstein, J.M., Franklin, M.J., Chandrasekaran, S., Deshpande, A., Hildrum, K., Madden, S., Raman, V., Shah, M.A.: Adaptive Query Processing: Technology in Evolution. IEEE Data Eng. Bull. **23**(2) (2000)
19. Intanagonwiwat, C., Estrin, D., Govindan, R., Heidermann, J.: Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In: ICDCS, 2002
20. Kempe, D., Dobra, A., Gehrke, J.: Gossip-Based Computation of Aggregate Information. In: FOCS, 2003
21. Kotidis, Y.: Snapshot Queries: Towards Data-Centric Sensor Networks. In: ICDE, 2005
22. Lazaridis, I., Mehrotra, S.: Approximate Selection Queries over Imprecise Data. In: ICDE, 2004
23. Lindsey, S., Raghavendra, C.S.: Pegasis: Power-Efficient Gathering in Sensor Information Systems. In: IEEE Aerospace Conference, 2002
24. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: A Tiny Aggregation Service for ad hoc Sensor Networks. In: OSDI Conference, 2002
25. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The Design of an Acquisitional Query processor for Sensor Networks. In: ACM SIGMOD, 2003
26. Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., Varma, R.: Query Processing, Resource Management, and Approximation in a Data Stream Management System. In: CIDR, 2003
27. Olston, C., Jiang, J., Widom, J.: Adaptive Filters for Continuous Queries over Distributed Data Streams. In: ACM SIGMOD, 2003
28. Olston, C., Loo, B.T., Widom, J.: Adaptive Precision Setting for Cached Approximate Value. In: SIGMOD, 2001
29. Olston, C., Widom, J.: Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In: VLDB, 2000
30. Olston, C., Widom, J.: Best-effort Cache Synchronization with Source Cooperation. In: ACM SIGMOD, 2002
31. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., Shenker, S.: GHT: A Geographic Hash Table for Data-Centric Storage. In: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, 2002
32. Sharaf, A., Beaver, J., Labrinidis, A., Chrysanthis, P.: Balancing Energy Efficiency and Quality of Aggregate Data in 1987 Sensor Networks. VLDB J., **13**(4), (2004)
33. Shnayder, V., Hempstead, M., Chen, B., Allen, G.W., Welsh, M.: Simulating the Power Consumption of Large-Scale Sensor Network Applications. In: Sensys, 2004
34. Singh, S., Woo, M., Raghavendra, C.S.: Power-aware routing in mobile ad hoc networks. In: ACM/IEEE International Conference on Mobile Computing and Networking, 1998
35. Soparkar, N., Silberschatz, A.: Data-value Partitioning and Virtual Messages. In: PODS, 1990
36. Tan, H.O., Korpeoglu, I.: Power Efficient Data Gathering and Aggregation in Wireless Sensor Networks. SIGMOD Rec., **32**(4), (2003)
37. Terry, D.B., Goldberg, D., Nichols, D., Oki, B.M.: Continuous Queries over Append-Only Databases. In: ACM SIGMOD, 1992
38. Yao, Y., Gehrke, J.: The Cougar Approach to In-Network Query Processing in Sensor Networks. SIGMOD Rec. **31**(3):9–18, 2002
39. Ye, W., Heidermann, J.: Medium Access Control in Wireless Sensor Networks. Technical report, USC/ISI, 2003