The VLDB Journal manuscript No.
(will be inserted by the editor)

Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks*

Mohamed A. Sharaf[1,2], **Jonathan Beaver**[1,3], **Alexandros Labrinidis**[1,4], **Panos K. Chrysanthis**[1,5]

[1] Department of Computer Science, University of Pittsburgh, Pittburgh, PA, 15260, USA.
[2] e-mail: msharaf@cs.pitt.edu
[3] e-mail: beaver@cs.pitt.edu
[4] e-mail: labrinid@cs.pitt.edu
[5] e-mail: panos@cs.pitt.edu

Received: TO BE COMPLETED / Revised version: TO BE COMPLETED

Abstract In-network aggregation has been proposed as one method for reducing energy consumption in sensor networks. In this paper, we explore two ideas in order to further reduce energy consumption in the context of in-network aggregation: First, *influencing the construction of the routing trees for sensor networks* with the goal of reducing the size of transmitted data. Toward this, we propose a *group-aware network configuration* method, that "clusters" along the same path sensor nodes that belong to the same group. Secondly, imposing a *hierarchy of output filters* on the sensor network with the goal of both reducing the size of transmitted data as well as minimizing the number of transmitted messages. More specifically, we propose a framework to use *temporal coherency tolerances* in conjunction with in-network aggregation to save energy at the sensor nodes while maintaining specified quality of data. These tolerances are based on user preferences or can be dictated by the network in cases where the network cannot support the current tolerance level. Our framework, called TiNA, works on top of existing in-network aggregation schemes. We evaluate experimentally our proposed schemes in the context of existing in-network aggregation schemes. We present experimental results measuring energy consumption, response time, as well as quality of data for Group By queries. Overall, our schemes provide significant energy savings with respect to communication and a negligible drop in quality of data.

1 Introduction

Advances in microelectronics have brought upon a new class of computing devices, that combine sensing instrumentation along with computation and communication capabilities, including having mini operating systems embedded in the sensor [14]. These sensor nodes, such as the Berkeley MICA Mote [13], are capable of collecting various measurements such as light, motion/acceleration, and temperature. Until recently, these sensor nodes where used within the confines of laboratories, factory equipments, or buildings. As they become smaller and cheaper, sensor nodes are expected to be ubiquitously deployed in the environment. This would enable them to collectively form large sensing networks over broad geographical areas.

Such large sensor networks can be used from monitoring endangered species [17, 23], to monitoring structural integrity of bridges [19], to patrolling borders. As such, sensor networks offer today an unprecedented level of interaction with the physical environment [6]. Within a few years, miniaturized, networked sensors have the potential to be embedded in all consumer devices, in all vehicles, or as part of continuous environmental monitoring. However, there are still many crucial problems with the deployment of such large sensor networks: limited storage, limited network bandwidth, poor inter-node communication, limited computational ability, and limited power of the sensor nodes. In this paper, we focus on the latter problem, i.e., reducing power consumption in sensor networks.

Several techniques have been proposed to alleviate the problem of limited power at the network level (such as energy-efficient routing and clustering [9, 36, 10]) and at the data management level (such as sampling [22], prediction [8], approximation [4], power-based query optimization [22], and data centric storage [28]). Another method at the data management level is in-network query processing (or aggregation). With in-network aggregation, a part of the computational work of the aggregation is performed within the sensor node before it sends the results out to the network. The reason why in-network aggregation reduces power consumption is that sensor power usage is dominated by transmission costs, as has been shown in [10, 16]. This can be easily illustrated by the following simple example of a sensor network used to monitor the average or the maximum temperature in a building (e.g., in order to quickly detect a fire or adjust the air conditioning). The default way to implement this is to have each sensor send its temperature reading up the network to the base

* This work is supported in part by NSF award ANI-0123705. The first author is supported in part by the Andrew Mellon Predoctoral Fellowship. This paper expands on the material presented in two workshops [31, 2].

station, with intermediate nodes responsible for just routing packets. In *in-network aggregation*, communication among sensor nodes is structured as a (routing) *tree* with the base station as its root. In this scheme, each node would incorporate its own reading together with the average computed so far by its children. As such, only one packet needs to be sent per node and each intermediate node computes the new average temperature before sending information further up the network. As a result, being able to transmit less data (because of aggregation instead of having to forward all the packets) will reduce energy consumption at the sensor nodes.

In this work we explore two ideas in order to further reduce energy consumption in the context of in-network aggregation:

1. *influencing the construction of the routing trees for sensor networks* with the goal of reducing the size of transmitted data; and
2. imposing a *hierarchy of output filters* on the sensor network with the goal of both reducing the size of transmitted data as well as minimizing the number of transmitted messages while ensuring a specified level of Quality of Data (QoD).

For our first idea, we consider the semantics of the query and the properties/attributes of the sensor nodes when configuring the sensor network (in addition to traditional shortest distance criteria). We have observed that the length of the messages sent by a node when processing Group-By queries depends on the number of groups existing in the routing subtree rooted at that node. This observation leads to the principle that reducing the number of groups considered at a sensor node (performing in-network aggregation) will reduce the size of the transmitted data and hence incur less energy cost for transmitting them. Based on this principle, our first contribution is a *group-aware network configuration* method, that "clusters" along the same path sensor nodes that belong to the same group.

For our second idea, we exploit temporal correlation in streams of sensor readings to suppress insignificant readings which can be tolerated. Further, suppressing such readings potentially allows nodes that do not have to transmit data to switch into *doze* or *sleep* mode, powering down their antennas. Doze mode offers the maximum possible saving in energy. Towards this, our second contribution is a framework, called TiNA (short for **T**emporal coherency-aware **i**n-**N**etwork **A**ggregation). TiNA works on top of existing in-network aggregation schemes such as TAG [21] and Cougar [35] and aims to balance the reduction in energy with the loss of QoD by adhering to user-specified QoD requirements. TiNA reduces energy consumption by using *temporal coherency tolerances* in conjunction with in-network aggregation to save energy at each sensor node, while maintaining the specified quality of data. These tolerances are based on user preferences or can be dictated by the network in cases where the network cannot support the current tolerance level.

In order to specify temporal coherency tolerance, TiNA introduces a new TOLERANCE clause in the SQL expression of sensor network queries. While it is the WHERE clause that acts as an *input* result filter, this new TOLERANCE clause acts as a hierarchical *output* transmission filter. By being a transmission filter, TiNA is able to save energy for two reasons. First, at the edge nodes (i.e., the leaf nodes of the routing tree), if a new reading falls inside the given tolerance the reading is not transmitted. Secondly, at the internal nodes, if aggregation eliminates values, transmitted messages have smaller size.

We have experimentally evaluated both our proposed schemes using simulation. Specifically, we have investigated the reduction in energy for group-aware network configuration for the sensor network implementations of TAG and Cougar, with and without utilizing TiNA. We have also studied the effect of TiNA on different Group-By and aggregation type queries, as well as how TiNA is affected by the rate that data changes. Additionally, we looked at TiNA's effects on the lifetime of the sensor network.

Our results show that our method, by not sending and by decreasing the size of messages, provides large gains in power savings over previous methods of in-network aggregation while minimizing the impact on quality of data. These results show that TiNA can reduce power consumption used by communication by up to 60% and extend the life of the sensor network by up to 270%. Furthermore, an additional 33% of energy used for communication can be saved by incorporating the group-aware network configuration with TiNA. Finally, our results also show that in some cases where the period to send is too short for all data to be propagated up through the network, TiNA increases the quality of data compared to existing in-network aggregation methods.

The rest of this paper is organized as follows. In Section 2 provides an overview of in-network aggregation. The idea of routing in sensor networks and an introduction to our new network configuration algorithm are presented in Section 3. We describe TiNA, our framework for temporal in-network aggregation in Section 4. In the same section, we describe how to implement TiNA on top of TAG and Cougar and in conjunction with our network configuration algorithm. Section 5 describes our simulation testbed, and then in Section 6 we show our experiments and results. We present related work in Section 7. We conclude in Section 8.

2 In-Network Aggregation

Directed diffusion [9, 16] is the prevailing data dissemination paradigm for sensor networks. In directed diffusion data generated by a sensor node is named using attribute-value pairs. A node requests data by sending interests for named data. Data matching the interest is then drawn towards the requesting node. Since data is self-identifying, this enables activation of application-specific caching, aggregation, and collaborative signal processing inside the network, which is collectively called *in-network processing*. Ad-hoc routing protocols (e.g., AODV[27]) can be used for request and data dissemination in sensor networks. These protocols, however, are end-to-end and will not allow for in-network processing. On the

contrary, in directed diffusion each sensor node is both a message source and a message sink at the same time. This enables a sensor node to seize a data packet that it is forwarding on behalf of another node, perform local, in-network processing on this packet, if applicable, and forward the newly generated packet up the path to the requesting node.

Cougar [3, 35] abstracted the named data generated by the sensor network as an append-only relational table. In this abstraction, an attribute in this table is either information about the sensor node (e.g., id, location) or data generated by this node (e.g., temperature, light). Queries that access this relational table can be either continuous [32, 22] or event-based [22]. In the former, the query result is updated periodically for a specified interval, while in the latter, the occurrence of an event triggers the data collection. Traditional ad-hoc queries that might be issued to probe present or historical data are also supported.

In the rest of this paper, we will only focus on continuous queries, more specifically, on aggregate continuous queries. Aggregate queries are particularly important in sensor networks where applications are often interested in summarized and consolidated data rather than detailed data [1]. For example, queries might be posed to periodically monitor the total occupancy of an office building, the maximum temperature in a volcanic area, or the average traffic density on a major road.

Continuous queries can be expressed using an extended SQL select statement:

```
SELECT {attributes, aggregates}
FROM sensors
WHERE conditions-A
GROUP BY {attributes}
HAVING conditions-B
EPOCH DURATION i | EVERY e
```

The first five clauses are the same as in standard SQL. We are focusing on standard SQL aggregation functions (AVG, SUM, MIN, MAX, COUNT). The two clauses in the sixth line were introduced by TAG [21] (the aggregation service for TinyDB [22]) and Cougar [35]. Clause *EPOCH DURATION i* was introduced by TAG. Parameter i is the epoch interval and it specifies the arrival rate of new results as required by the user. Hence, once every *epoch*, the user is expecting the network to produce a new answer to the posed continuous query. Clause *EVERY e*, was introduced by Cougar and, similarly to *EPOCH DURATION*, specifies the interval between two consecutive results (which is called a *round*). In this paper, we will be using both terms interchangeably.

Communication in a sensor network can be viewed as a tree, with the root being the base station. Synchronizing the transmission between nodes on a single path to the root is crucial for efficient in-network aggregation. A sensor (parent) needs to wait until it receives data from all nodes routing through it (children) before reporting its own reading. This delay is needed so that the parent node p can combine the partial aggregates reported by its children with its own reading and then send one message representing the partial aggregation of values sampled at the subtree rooted at p. The problem of deciding how long to wait is treated differently in Cougar and TinyDB. The details of the synchronization mechanisms of these two systems are discussed in greater detail below.

Synchronization in TAG Synchronization in TAG is accomplished by making a parent node wait for a certain time interval before reporting its own reading. Specifically, TAG subdivides the *EPOCH DURATION*, specified by the user in the SQL statement, into shorter intervals called communication slots. The number of these slots is equal to the maximum depth of the routing tree (d). The duration of each communication slot is *(EPOCH DURATION) /d*.

During a given communication slot, there will be one level of the tree sending and one level listening. In the following slot, those that were sending will go into doze mode until the next epoch, while the nodes that were receiving will now be transmitting. The cycle continues until all levels have sent their readings to their parents. When a parent receives the information, it aggregates the information of all children along with its own readings before sending the aggregate further up the tree. This synchronization scheme provides a query result every epoch duration. TAG also proposes pipelining, in which a node waits several epochs of gathering data before sending it up the tree, in order to allow increased sampling rates beyond the rate allowed by the communication slot scheme. In this paper, however, when we refer to TAG we will be referring to only the basic slot based version of synchronization in TAG.

Synchronization in Cougar Cougar takes a pragmatic approach to synchronization. This approach is motivated by the fact that for a long running query, the communication pattern between two sensors is consistent over short periods of time. Hence, in a certain round, if node *p* receives data from a node *c*, then node *p* will realize it is the parent of node *c*. Node *p* will add *c* to its *waiting list* and predict to hear from it in subsequent rounds. In the following rounds, node *p* will not report its reading until it hears from all the nodes on its waiting list. In order to prevent node *p* from waiting on *c* indefinitely, node *c* transmits either a reading or a notification packet during each round. The notification packet indicates that the current reading at *c* does not satisfy the predicate associated with the query.

Since a parent only has to wait to hear from its children, this results in messages from different parts of the tree going up the tree at different rates. In order to accommodate this, a parent must listen the entire time from the start of a round until it hears from all its children. This ensures it hears from every child and does not miss one by not listening from the beginning. This synchronization scheme adapts to network dynamics. Each node delivers its partial result as soon as it is available, which should also reduce response time for uncongested networks.

[1] See [12] for a discussion about more sophisticated applications.

3 Energy Efficient Data Routing in Sensor Networks

The ability to route data from the various nodes of the sensor network towards a central sink point (i.e., the base station) is fundamental to the operation of sensor networks. To support routing of data, the sensor network can be configured into some form of hierarchy, such as routing trees [9,21] or clustering [36,10]. The most commonly used routing scheme for in-network aggregation is in the form of a tree, where each node (child) selects a *gradient* [9] or *parent* [21] to propagate its own readings.

The sensor network constructs the routing tree along with the propagation of the query. The construction of the routing tree is initiated from the base station. We assume that a new query in our model originates at the base station which forwards it to the nearest sensor node. This sensor node will then be in charge of disseminating the query down to all the sensor nodes in the network and to gather the results back from all the sensor nodes.

There are several ways in which the routing tree can be built. One relatively simple way is to try to create the tree in such a way that the distance between any two nodes is minimized. This can be done in a greedy manner [15] by having the first node heard from chosen as the parent. The intuition behind this choice is the assumption that if a node is heard from first, it was most likely the closest to the child. We call this protocol First-Heard-From in this paper. For simplicity we will use this protocol as a comparison basis with our new protocol Group-Aware Network Configuration which will build on the basis laid by the First-Heard-From protocol.

It should be noted that in an environment which is lossy in nature, using the First-Heard-From protocol in general is not appropriate. As pointed out in [33], packet-loss rate of a channel and distance are not directly related. A node which is closer may actually have a higher loss rate than a node which is further away, leading to more retransmissions and consequently, more energy being used. In such an environment a more appropriate protocol would be the one that considers loss rates and uses this in choosing the best parent.

3.1 First-Heard-From Network Configuration

The basic idea behind the First-Heard-From (FHF) network configuration algorithm is as follows. Starting from the root node, nodes transmit the new query. Children nodes will select as their parent the first node they hear from and continue the process by further propagating the new query to all neighboring nodes. The process terminates when all nodes have been "connected" via the routing tree.

The FHF method is formally described as follows:

1. The root sensor prepares a query message which includes the query specification. The root sensor also sets the (L_s) value in the message to its level value (i.e., L_{root} which is 0 initially). It then broadcasts this query message to the neighboring sensors.
2. A sensor i that receives a query message and has its level value currently equal to ∞ will set its level to the level of the node it heard from, plus one. That is, $L_i = L_s + 1$.
3. Sensor i will also set its parent value P_i to Id_s. It then will set Id_s and L_s in the query message to its own Id_i and L_i respectively and broadcast the query message to its neighbors.
4. Steps 2 and 3 are repeated until every node i in the network receives a copy of the query message and is assigned a level L_i and a parent P_i.

In cases where nodes that are adjacent are equally distant from one another, [12] suggests that a node will uniformly and at random select from the available parents. This uniform selection will help in balancing the load among different nodes that are equally adequate as parents. However, the main weakness of this method is that it in fact creates the network in a random way (only based on network proximity). The children assign parents based on whichever node happened to broadcast the routing message first. This method as well as other similar methods that consider only the network characteristics, such as link low-loss rate, fail to consider the semantics of the query or the properties/attributes of the sensor nodes and hence cannot take any opportunities for energy savings.

3.2 Group-Aware Network Configuration

In order to have a network configuration method that considers the semantics of the query and the properties of the sensor nodes, we look closely at how in-network aggregation works. In-network aggregation will depend on the query *attributes* and the *aggregation function*. On the one hand, the list of attributes in the Group-By clause subdivides the query result into a set of groups. The number of these groups is equal to the number of combinations of distinct values for the list of attributes. Two readings from two different sensor nodes are only aggregated together if they belong to the same group. On the other hand, the aggregation function determines the structure of the partial aggregate and the partial aggregation process. For example, consider the case where the aggregate function is SUM. In this case, the partial aggregate generated by a routing sensor node is simply the sum of all readings that are forwarded through this sensor node. However, if the aggregate function is AVERAGE, then each routing sensor node will generate a partial aggregate that consists of the sum of the readings and their count. Eventually, the root sensor node will use the sum and count to compute the average value for each group before forwarding it to the base station for further processing and dissemination.

Because aggregation combines all the readings for a particular group into one group aggregate reading, creating a routing tree that keeps members of the same group within the same path in the routing tree should help decrease the energy used. The reason is simple: by *"clustering" along the same path* nodes that belong to the same group, the messages sent from these nodes will contain less groups (i.e., be shorter,

thus reducing communication costs). With this intuition in mind, we introduce our *Group-Aware Network Configuration* method, or *GaNC* for short.

The way is which the GaNC algorithm constructs the routing tree is as follows:

1. The root sensor prepares a query message which includes the query specification. The root sensor also sets the (L_s) value in the message to its level value (i.e., L_{root}, which is initially set to 0). It also sets the (G_s) to be its group id. It then broadcasts this query message to the neighboring sensors.
2. A sensor i that receives a query message and has its level value currently equal to ∞ will set its level to the level of the node it heard from, plus one. That is, $L_i = L_s + 1$.
3. Sensor i will also set its parent value P_i to Id_s and its parent's group id PG_i to G_s. It will then set Id_s, L_s and G_s in the query message to its own Id_i, L_i and G_i respectively and broadcast the query message to its neighbors.
4. While there are still query messages being propagated around the network, node i continues to listen to all messages it can hear.
5. If node i hears a message from a node at the same level as itself minus one ($L_i - 1$), it uses *tie-breaker conditions* to decide if this new node should become its new parent. If so, node i makes Id_s its new parent.
6. Steps 2-5 are repeated until all query messages in the network have been sent out and received.

The GaNC algorithm is built on the basis of the FHF protocol. The main difference is that a child under the GaNC method can switch to a better parent while the tree is still being built. This switch is based on a set of *tie-breaker conditions* that go beyond the network characteristics, to introduce the semantics of aggregation. GaNC works with semantic groups which are static in nature and are based on hard coded sensor attributes, such as floor number or location.

The goal of the group-aware network configuration algorithm is to incorporate group identity into the routing tree construction. As such, the first tie-breaker condition (for Step 5 of the algorithm) is whether the child has the same group id as the parent. As long as a child is within listening distance of multiple parent choices, a child will choose a parent that has the same group id as itself instead of a parent from a different group. This is a choice that will allow parents and children to be in the same group as much as possible.

In the general case, a sensor node will be within listening range of multiple other nodes. Despite the savings in clustering nodes of the same group along the same path, a node that is far away will require significantly more transmission energy, and as such is not a good candidate. For that reason, we introduce a *distance factor*, df, that will limit the maximum range for which we consider candidate sensor nodes (for coming up with a "better" parent node). Under this approach, if d_i is the shortest distance seen so far (based on an estimation from signal strength), we will only consider nodes whose distance from a child node is at most $df \times d_i$, for example, for $df = 1.2$ we will only allow up to 20% more than the minimum distance.

Thus, the second tie-breaker is the estimated distance from the child to the parent. The parent with the lowest distance will be chosen in cases when there is more than one parent to choose from (that is in the same group as the child), or when no parents are in the same group as the child. The reason for this is that in both cases, routing through the closest parent will save transmission energy for the child.

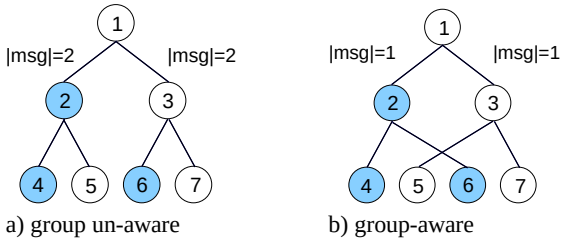

Fig. 1 Benefi ts of group-aware network confi guration

To better illustrate the basic motivation and benefit of GaNC, we use the simple example shown in Figure 1. In this figure, nodes 2, 4, and 6 (the shaded ones) belong to one group, whereas nodes 1, 3, 5, and 7 belong to a different group. Let us assume that under the standard FHF network configuration (Figure 1a), nodes 4 and 5 pick 2 as their parent, whereas nodes 6 and 7 pick 3 as their parent. Using in-network aggregation, the message sizes from nodes 2 and 3 to the root of the network will both be 2 (i.e., contain partial aggregates from two groups). On the other hand, if we cluster along the same path nodes that belong to the same group (Figure 1b) we reduce the size of messages from nodes 2 and 3 in half: each message will only contain the partial aggregate from a single group.

Overall, this algorithm uses more information to create a better routing tree for aggregation type queries. The idea of using the group ids to put children with parents should decrease energy consumption by decreasing message sizes: less groups are being sent from a single parent, since its children are in the same group as itself. In addition, nodes are transmitting over shorter distances since distance is being used as a secondary tie-breaker criterion. This in general will decrease the transmission energy used, which increases exponentially with distance. These two effects should decrease the overall energy consumed by the network and make the routing tree more efficient with aggregation queries.

4 Temporal Coherency-Aware In-Network Aggregation

In this section, we focus on the actual in-network aggregation on top of a routing tree. Specifically, we present TiNA (short for **T**emporal coherency-aware **i**n-**N**etwork **A**ggregation) that exploits the temporal correlation in a sequence of sensor readings to reduce energy consumption by suppressing readings that do not affect the expected quality of data as defined by the user or application.

TiNA is built as a layer that operates on top of in-network aggregation systems in order to minimize energy consumption throughout the entire sensor network. In designing TiNA we considered the different features of the existing in-network aggregation systems, namely, Cougar and TAG. As discussed above, in these systems all sensor readings are passed up the tree once per epoch or round, as defined in the query. Aggregation is done at the internal nodes while information is forwarded up the routing tree.

The contribution of TiNA comes in how it selectively decides what information to forward up the routing tree by applying a hierarchy of filters along each path of the network. To perform data filtering in a controlled, well-defined manner, TiNA introduces the *TOLERANCE* clause in the query specification as follows:

```
SELECT {attributes, aggregates}   ┐
FROM sensors                      │
WHERE conditions-A                │  Standard SQL
GROUP BY {attributes}             │
HAVING conditions-B               ┘
EPOCH DURATION i | EVERY e        ]  TAG and Cougar
TOLERANCE tct                     ]  Introduced by TiNA
```

The *tct* parameter of the TOLERANCE clause is used by the user to specify the temporal coherency tolerance for the query. The *tct* value acts as an output filter at the readings level, suppressing readings within the range specified by *tct*. For example, if the user specifies *tct* = 10%, the sensor network will only report sensor readings that differ from the previously reported readings by more than 10%. Values for *tct* range from 0, which indicates to report readings if any change occurs, to any positive number. This *tct* is the maximum change that can occur to the overall quality of data in the system using TiNA.

In this paper, we use a relative definition of *tct*, although a similar absolute formulation of *tct* is possible. However, the relative formulation of *tct* allows a uniform, easy to understand definition of user tolerance on heterogeneous data sources, where the domain of sensed values is different from one sensor to another. This is especially important when all readings and groups in the aggregated result are equally important.

A TiNA sensor node must keep additional information in order to utilize the temporal coherency tolerance. The information kept at a certain sensor depends on its position in the routing tree (i.e., a leaf or an internal node). Leaf nodes keep only the *last reported reading* which is defined as the last reading successfully sent by a sensor to its parent. Internal nodes, in addition to the last reported reading for that node, keep the last reported data it received from each child. This data can either be a simple reading reported by a leaf node or a partial result reported by an internal node.

– At a leaf node, when a new reading of value V_{new} is available, this new value is compared against the last reported reading (say V_{old}). The new value is reported iff $\frac{|V_{new}-V_{old}|}{V_{new}} > tct$ otherwise the value V_{new} is suppressed.

– At an internal (i.e., parent) node, the following sequence of operations takes place when the node is listening and aggregating data:

1. It collects the data reported by its children. If a parent does not receive complete data from any of its children, it fills in the missing data using some or all of the last reported data from that child.
2. It combines the complete data for each child together to compute the partial result for the subtree rooted at this parent node.
3. The internal node then takes its own reading. If its own reading can be aggregated within a group that already exists in the partial result, then the reading is aggregated regardless of its *tct* value. By doing this, TiNA is taking advantage of the in-network aggregation mechanism where the aggregation at the parent will improve the accuracy of the query results without increasing the size of the partial result.
4. If including the new reading would result in creating a new group, then the reading is only added if it violates the *tct* value, otherwise it is suppressed in order to minimize the partial result size while still maintaining the specified tolerance.
5. Finally, the internal node takes an old partial result (one computed from all children's old data and its own old reading) and compares it against the new partial result it has created. For any tuple where the partial aggregate value has not changed, that tuple is eliminated from the final partial result. This elimination is equivalent to applying *tct*=0 at the partial aggregate level which serves as an upper level filter.

Having the last operation repeated at every parent node along all the network paths provides a hierarchy of filters on every path. Setting the *tct* to zero for the hierarchical filtering at intermediate nodes ensures that the partial aggregates, and eventually the final aggregates, are always within the user-specified tct.

This hierarchy of filters is important for the incremental processing of aggregate queries as it captures cases of temporal correlation that cannot be captured at the readings level by individual sensors. For example, consider the SUM aggregation function; readings from different sensors might change from one round to another, however, it is possible that the overall sum stays the same. This can only be detected at a parent node which intercepts the stream of readings generated by these sensors and acts as an intermediate *centralized* stream processor. Note that this operation can provide a completely empty partial result or a partial result that is missing few groups compared to the old partial result. In both cases, this node relies on the fact that its parent stored its last reported data and it will use it to supply the missing groups as mentioned in step 2.

Since parent nodes store the last reported reading of their children, supporting the WHERE clause and handling node failures becomes a non trivial issue. In both cases, the parent needs to know whether the stored readings from a child

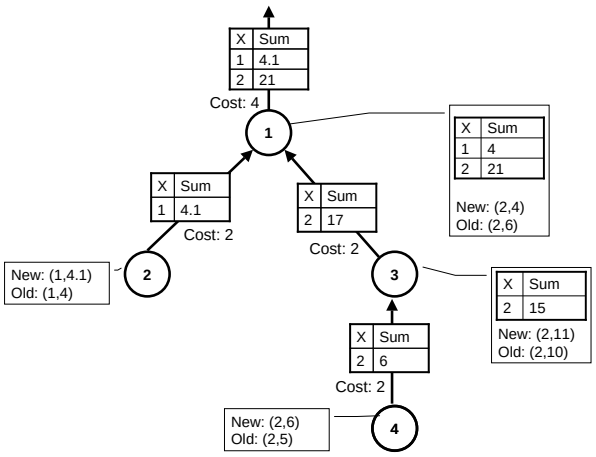

Fig. 2 Query using TAG

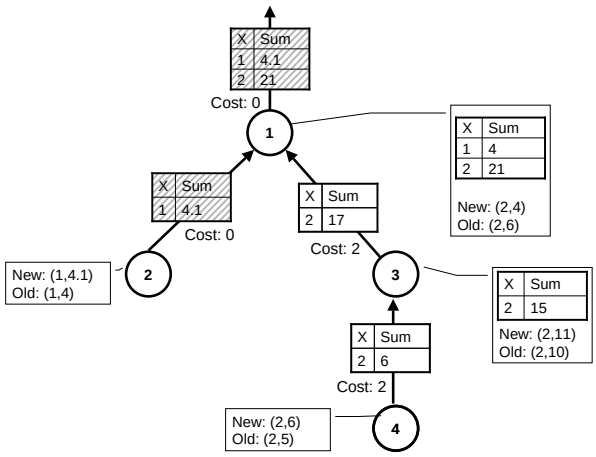

Fig. 3 Query using TiNA on top of TAG

are still valid. Specifically, the issue that arises when using the WHERE clause and having fault tolerance is how to determine the difference between a) a value not meeting the conditions of the WHERE clause, b) a value not being sent because it is inside the given tolerance range and c) a node dying and thus not being in the network anymore. One simple way to deal with these cases is to adopt the notion of a notification packet from Cougar and expand its usage to support TiNA functionality. The notification packet is simply a packet containing a notification bit.

To handle the WHERE clause, a child uses the notification packet with the notification bit turned off as a way to invalidate stored readings at its parent. When a parent receives a notification packet from a child with a notification bit set to 0(off), it knows that the reading it has stored for that child is invalid and cannot be used in future aggregations. It will not substitute any readings for that child until the child supplies it with a new data. The notification bit could also be appended into partial data in the case of internal nodes to allow parents to know which group values to stop using as well.

To handle node failures, children are required to send *heartbeat* messages to their parents at regular intervals while they are suppressing values. This heartbeat is simple a notification packet with the notification bit set to 1(on). This lets the parent know the child is still alive and its reading is still valid and should be used to substitute in aggregations. If a parent does not receive a heartbeat message from a child after a certain period of time, the parent will mark the child as disconnected until it hears from the child again. This heartbeat technique will also work for mobile sensors, where a sensor might change its location in the network, thus switching parents.

In the case of parent node failures, we utilize the periodic network reconfiguration to reconnect the children back to the sensor network (through a new parent). In these cases, the children nodes must first transmit their new value regardless of the *tct*, as the new parent will not have any of the old state information.

In the remaining of this section, we describe how TiNA works on top of TAG and on top of Cougar. We also describe the synergy between TiNA and GaNC.

4.1 TiNA on top of TAG

TiNA works on top of TAG by taking advantage of the predefined sending and receiving communication slots in each epoch. During a given communication slot, all children would usually be sending their readings with the parents listening. In TiNA, the parents are still listening, but it may be the case that a child does not send a reading. In this case, when the communication slot expires, the parent checks to see if it heard from all its children. For each child it did not hear from, the parent takes the last reported data it has for that child and aggregates it with data from the other children. The rest of the scheme then operates as usual for TiNA.

Example execution of TiNA Figures 2 and 3 show a comparison between two in-network aggregation schemes during one epoch. Figure 2 shows the execution of the TAG base case, whereas Figure 3 shows the execution of TiNA on top of TAG. We assume a query to get the total light for all rooms, while grouping by floor, with the *tct* = 10%:

```
SELECT {FLOOR, SUM(LIGHT)}
FROM SENSORS
GROUP BY {FLOOR}
EPOCH DURATION 30s
TOLERANCE 10%
```

Nodes are represented as circles in the figures and the flow of data from child to parent is represented with arrows. The boxes connected to each node represent the current state at the node. The current state consists of the last reported reading, called *Old*, the current reading, called *New*, and a table representing that node's previously reported partial result. As an example, consider node 3, its *Old* reading is 10, its *New* reading is 11, and its previously reported partial result is (2,15). In this partial result, the value 2 is the group identifier

(e.g., the x-coordinate), and 15 is the previous partial aggregate value generated by summing its own old reading (=10) to the old reading reported by node 4 (=5).

Tables along the connection lines represent the data being sent from child to parent. The *Cost* number under each of these tables is the cost to send this data from child to parent. The cost is just the size of the table, since it is used to illustrate relative costs. For example, a table with cost 4 is twice as big as a table with cost 2, and thus will need to send twice as much information, resulting in twice the transmission cost.

In Figure 2, we see a one-epoch execution of the query using TAG without TiNA. In this example, every reading is sent from child to parent. We illustrate the benefit of using TiNA on top of TAG in Figure 3. The setup is the same as before, but now the sensor network has employed the TiNA scheme. Savings from TiNA (parts of messages that do not need to be sent) are shown by being shaded out. Note that in the epoch demonstrated, heartbeat messages are not needed because all old values were sent in the previous epoch.

When it is time for node 4 to send, it first checks its new reading against its old reading. Since the new reading (=6) differs by more than 10% from its old reading (=5), it will send its new reading and replace the old one. During the next communication slot, nodes 2 and 3 check their readings against their last readings. For both nodes, the change is less than 10%. However, node 2 will suppress transmission, while node 3 will aggregate its new reading with the reading reported from node 4 since they both belong to the same group. As explained above, this operation improves the result accuracy at no extra transmission costs. In the final slot, node 1 will aggregate its own reading into the partial results it received from node 3. However, for group 2, the new partial aggregate value (=21) happened to be equal to the one previously reported by node 1. Thus, the partial aggregate for group 2 is suppressed and node 1 will not transmit any messages for this epoch.

As shown in the example execution of TiNA above, sensor nodes save significant energy by suppressing transmission at the reading level and the partial aggregate level. These savings are propagated up the levels of the routing tree. In our example, the sum of sizes of all messages under the plain TAG approach was 10, which was reduced to 4 when using TiNA. Such a 60% decrease in the transmission cost reduces energy consumption dramatically, but comes with a slight decrease in the accuracy of the results (4.0 instead of 4.1 in one of the groups).

4.2 TiNA on top of Cougar

Using TiNA on top of Cougar does not involve any change to the underlying Cougar framework. However, implementing TiNA on top of Cougar is a little different than implementing TiNA on top of TAG. In Cougar, parents wait to hear from all their children before aggregating their own reading and sending the result further up the routing tree.

Using TiNA on top of Cougar, TiNA does not suppress the notification messages of Cougar but it replaces them with those of TiNA. Thus, when a child would choose not to send a reading because the reading did not violate the given *tct*, it would send a heartbeat message (i.e., a TiNA notification packet with the notification bit on). This will inform the parent that the stored reading of that child is still valid and that the parent can proceed with its processing.

It should be noted that the size of the notification packet is typically one bit in addition to the header, which allows for significant savings when compared to the size of a data packet that contains a reading. These savings are even more noticeable if that heartbeat is sent instead of transmitting a partial result that consists of multiple aggregates that are all within the *tct*.

We illustrate how TiNA works on top of Cougar using Figures 2 and 3 from the TAG example. Instead of each sensor sending during its given time slice (in TAG), a sensor will send once it hears from all its children (in Cougar). Furthermore, in Figure 3 nodes 2 and 1 will transmit a heartbeat instead of sending nothing. The rest of the figure is exactly the same, with the group data that is not sent in TiNA on top of TAG, not being sent in TiNA on top of Cougar either. As long as the parent hears from the child, it can fill in any information that is necessary. Finally, note that in the case of TiNA on top of Cougar the savings would come not only from the decrease in transmission size, but also from the decrease in response time. Since a parent has to listen until it hears from all its children, by using TiNA, the time a parent hears from all its children is also decreased, thus increasing the overall savings.

4.3 TiNA with Group-Aware Network Configuration

TiNA can further improve its energy efficiency when it is deployed together with a group-aware network configuration method (GaNC). TiNA suppresses values at internal nodes based on the aggregate values of each group. The more values from a single group the more chances that changes in these values will counteract each-other and as such, transmission of the group will be suppressed.

There is an added benefit to using GaNC that is unique to TiNA and can improve the overall quality of data. Remember that TiNA will “piggy-back” new readings of an internal node regardless of the amount of change, if the internal node’s group was going to be transmitted because of children input. In this optimization, the energy remains the same since adding an extra item into the aggregation of a group, does not increase the message size. In the case of GaNC, by clustering along the same path to the root nodes that belong to the same group, we increase the probability that a parent node matches the group of its children and thus include its own reading (i.e., get a “free ride”). This is expected to increase the overall quality of data without increasing energy consumption.

Let us now illustrate how TiNA can further boost the energy savings that are gained from using GaNC. Consider for example a network of nodes deployed within a building keeping track of the number of people in the building, grouped by

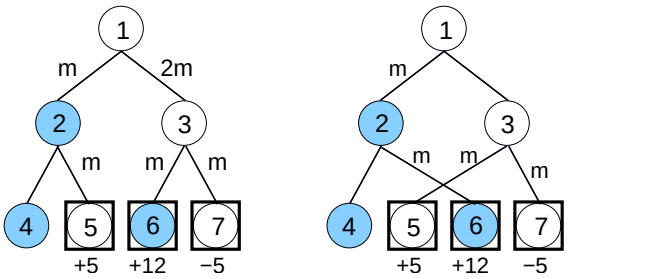

Fig. 4 Benefi ts of group-aware network confi guration with TiNA. Nodes in boxes indicate change in value more than tct (the value difference is listed underneath). m indicates transmission of a message of unit size.

floors. Figure 4a shows a section of the tree in this building's network as it would be built using the FHF network configuration method and Figure 4b shows the same section of the tree using GaNC instead. These trees are the same as in Figure 1. The nodes are shown as circles and the groups they belong to are based on the color (there are two groups, shaded (S) and not shaded (NS)). Finally, nodes in boxes indicate that their value changed by more than the *tct* and thus need to report the new values (the value difference in number of people is listed underneath).

In Figure 4a, normal TiNA execution would show a total of 5 messages being sent. The three messages from children to parents at the lowest level would be of size 1 (1 group with value) and the messages from the second level to the top level would be of size 1 and 2. This leads to total message size of 6. In addition, we have node 2 changing by less than the tct, so its reading is not sent unless its group is already being sent up; the remainder of the nodes are unchanged in value. Finally, nodes 5 and 7 at the lowest level change by complimentary amounts (5 people left that room, node 7, and moved to the other room, node 5).

If instead of the configuration in Figure 4a we use the new configuration shown in Figure 4b, the benefits of GaNC are more apparent. In this case, we first get the benefit that nodes 5 and 7 are routed through the same parent, so their complementary data changes cancel each other. This makes sense because the number of people on the floor did not change, they just moved their meeting to another room. This will save the transmission from node 3 to node 1. In addition, since node 2 now has a child in its own group that is sending through it, it can add its own reading into the message, thus increasing the quality of data for the query. By configuring the tree in a group-aware fashion, the total size of all messages was reduced to 4. In addition, the overall quality of data has increased, because more nodes are sending more precise information. This shows that by using TiNA with GaNC, the overall energy used in the network can be decreased and the overall quality of data can be increased.

5 Evaluation Testbed

In order to study the effects of the proposed TiNA framework, we created a simulation environment using CSIM [29]. Following typical sensor network simulation practices, the simulated network was configured as a grid of sensors. We assumed a lossless communication environment in which each node could transmit data to sensor nodes that were at most one hop away from it. In a grid this means it could only transmit to at most 8 other nodes. When experimenting with the GaNC protocol we will only use group semantics for network configuration (the first tiebreaker) since the distance tiebreaker (the second tiebreaker) is not applicable in such a grid setting.

We simulated a simplified version of the CSMA/CA contention-based MAC protocol, where a sender node will perform a channel sensing before initiating a transmission. If a node fails to get the medium, it goes to sleep for a random backoff period and then starts sensing the channel again. The node will only transmit its message when the channel is free. Before transmitting the message, the sender node will send a Request to Send (RTS) packet to the destination which replies with a Clear to Send (CTS) packet. These packets are used to ensure that other nodes in the destination transmission range will seize transmission so that collision will not occur. In our CSMA/CA implementation, we do not explicitly include the extra costs for the RTS and CTS packets, however, we are studying the impact of the communication protocol overhead in Section 6.7.

Using the simulator, we performed extensive experiments to evaluate the performance of TiNA, when used on top of TAG or on top of Cougar. For fairness in comparisons, we simulated the optimized version of TAG where a *child cache* is used [21]. In this caching scheme, a parent stores the partial aggregates reported by its children, and uses those aggregates when new ones are not available. This scheme is particularly important in the cases where communication is unreliable or the epoch duration is too short.

In the following experiments, the Group-By query is of the form described in Section 3.2, and the sensor network produces a result at intervals defined in the query. The objective of the query is to aggregate a measure (e.g., temperature) across different regions of the network. In the default case, the set of *attributes* used by the SELECT and the GROUP BY clauses is any valid combination of the sensors' X and Y coordinates, hence, *attributes* = {}, {X}, {Y}, or {X,Y}. For example, a query where *attributes* = X subdivides the sensors readings into a number of groups equal to the number of possible values of X (i.e., the width of the grid). In the answer for this query, readings from all sensors that have the same X coordinate are aggregated together according to the aggregate function. For our experiments, we focused on the standard SQL aggregation functions SUM, AVERAGE, and MAX. We did not include the MIN function, which is similar to MAX, nor did we include COUNT, which is similar to SUM.

We summarize all the experiment parameters in Table 1. Next, we will present the models we used for generating our synthesized data, whereas an analysis of real data is deferred to Section 6.4.

Parameter	Value	Default
Grid Size	11x11 – 45x45	15x15
Packet Header Size	2 - 8 bytes	4
Aggregate	MAX, SUM, AVG	SUM
Number of Attributes	0 – 2	1
Epoch Durations	240 – 7404 mSec	1408 mSec
tct	0% – 30%	
Number of Epochs	100	
In-network Aggregation	Cougar, TAG	Cougar
Routing Scheme	FHF, GaNC	FHF
Randomness Degree	0.0 – 1.0	0.5
Random Step Size Limit	5%–25% of domain	5%
Peak Amplitude(A)	50	
Period(T)	20 – 60	20
Phase(ϕ)	0°–180°	0°

Table 1 Simulation Parameters

5.1 Random Walk

One model we used for data generation is the random walk model. In this model, the domain of values was between 1 and 100 (to approximate temperature readings in Fahrenheit). A sensor reading is generated once at the beginning of each query interval. The value changes between one interval to the next with a probability known as the *randomness degree* (*RD*). Each time a sample is to be generated, a coin is tossed. If the coin value is less than RD, then a new value is generated, otherwise the sample value will be the same as before. For example, if $RD = 0.0$, then the value sampled by a sensor will never change, while if $RD = 0.5$, then there is a 50% chance that the new value at time *t* is different from the value at time *t+1*. We used the *Random Step Size Limit* to restrict how much the new value can deviate from the previous value. This limit is expressed as a percentage over the domain of values. In our case, a 5% step size limit implies that a new temperature reading can be at most 5 degrees less/more than the previous reading.

5.2 Sinusoidal Data

In the sinusoidal data model, each sensor generates a stream of values similar to a sinusoidal signal of the form:

$f(t) = A\sin(\frac{t}{T}2\pi + \phi)$

where A is the peak amplitude, t is the current round/epoch, T is the period in terms of number of epochs, and ϕ is the phase. We set the value of A to 50 and we also set $f(t) = f(t) + A + 1$ at any time t in order to retain the same domain of values provided by the random walk model.

Using this model, we can control generating data streams with different properties. One such property is the temporal correlation between values sensed at different sensors at the same instance. This correlation can be controlled using the ϕ parameter which enables us to set the relative direction of changes in values. This correlation is especially important for aggregate functions where the interaction between values can lead to a constructive or destructive aggregation. A destructive aggregation is when values are changing in different directions but the net results does not change significantly. A constructive aggregation is when values change in the same direction so that the net aggregate significantly changes.

5.3 Performance Metrics

In our experiments we focused on three measurements: energy consumption for communication, relative error, and response time.

Energy: Energy is consumed in four main activities in sensor networks: transmitting, listening, processing, and sampling. We focused on transmission and listening power, since the amount of time spent sampling is the same for all techniques. We did not include energy required for processing because it is negligible compared to that needed for communication.

As mentioned before, a sensor node will send its data to the root through its assigned parent. A parent node is one hop away from its child, and one hop closer to the root than its child. So every node sends its data exactly one hop away, all of which are the same distance from one another. This allows us to assume a uniform cost of transmitting data. However, the overall energy consumed to transmit a partial result depends on the size of the partial results and the number of messages.

The values of the parameters needed to calculate the transmission cost were the same as in [13]. Specifically, we simulated sensors operating at 3 Volts and capable of transmitting data at a rate of 40 Kbps. The transmit current draw is 0.012 Amp while the receive current is 0.0018. Hence, the cost of transmitting one bit in terms of energy consumption units ($Joules$) is computed as:

T_{cost} = 3 Volt * 0.012 Amp * 1/40,000 Sec = 0.9 μJoules.

The cost of listening for one second is computed as:

R_{cost} = 3 Volt * 0.0018 Amp = 0.0054 Joules.

The energy consumed during listening is independent of the number of messages received by the sensor. It only depends on the time spent by the sensor being active and listening. In TAG that time is specified by the communication slot interval. Cougar does not specify when a sensor stops listening and switches to doze mode. Hence, in our simulation, we assumed that each sensor will start listening at the beginning of each round. After a sensor receives data from all nodes on its waiting list it will switch to doze mode.

Relative Error Metric: The *relative error metric* (REM) is a measure of how close the exact answer and the approximate

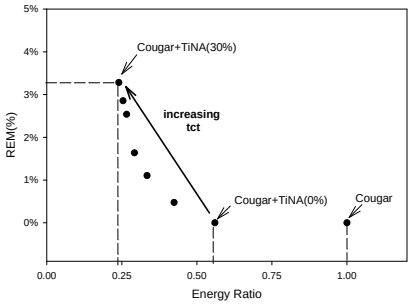

Fig. 5 Relative Error Metric (REM) of TiNA on top of Cougar, where $Energy\ Ratio = \frac{Energy\ using\ TiNA}{Energy\ using\ Cougar}$

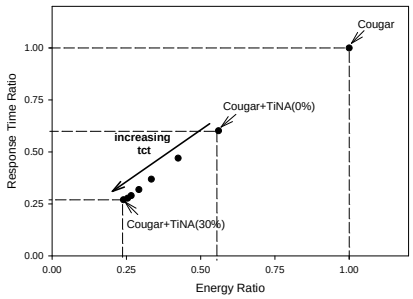

Fig. 6 Response time of TiNA on top of Cougar, where $Response\ \ Time\ \ Ratio = \frac{Response\ Time\ using\ TiNA}{Response\ Time\ using\ Cougar}$

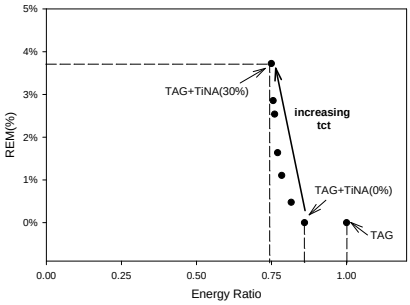

Fig. 7 Relative Error Metric (REM) of TiNA on top of TAG, where $Energy\ Ratio = \frac{Energy\ using\ TiNA}{Energy\ using\ TAG}$

answer are. The exact answer is generated if all sensors deliver their current readings within the epoch/round time. An approximate answer is one where some sensors fail to send their current reading or decide not to send it. A sensor fails to report a reading because of network congestion or short epoch interval. A sensor decides not to send a message because of the user-specified temporal coherency tolerance.

In order to compute REM, we first need to measure the error over the Group-By query. We measured this error as described in [1]. Assume a query aggregates over a measure attribute M. Let $\{g_1, ..., g_n\}$ be the set of all groups in the exact answer to the query. Finally, let m_i and m_i' be the exact and approximate aggregate values over M in the group g_i. Then, the error ϵ_i in group g_i is defined to be the relative error, i.e., $\epsilon_i = \frac{(|m_i - m_i'|)}{m_i}$. The error δ over the Group-By query is defined as: $\delta = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i$ finally, the average REM (or simply, REM) over time is defined as:

$$REM = \frac{1}{T}\sum_{t=1}^{T} \delta_t$$

where δ_t is the query error at $epoch_t$.

We are using REM as indication of the *quality of data* (QoD), where a high REM reflects a low QoD, while a low REM corresponds to a high QoD. Hence, we will be using both terms interchangeably.

Response Time: The *response time* is a metric that applies only to Cougar. As we mentioned earlier, the Cougar synchronization method allows results to be delivered before the end of the round interval, especially in the case of a lightly loaded network. In the case of TAG, the response time is always equal to the epoch duration.

6 Experiments and Results

6.1 Sensitivity to temporal coherency tolerance

In the first experiment, we vary the *tct* value and measure its effect on the relative error metric (REM), response time, and energy for TiNA versus the base cases for Cougar and TAG. The experiment uses the default values from Table 1 except for the *tct*, which we varied from 0% to 30%. Unless otherwise noted, the sensor network in all experiments was configured using the FHF method (as described in Section 3).

Figure 5 shows the trade-off between energy savings and relative error when using TiNA on top of Cougar. We have the ratio of energy required compared to that needed by plain Cougar on the X-axis, and the relative error (which is 0% for Cougar) on the Y-axis. For perfect QoD, and thus no relative error (*tct=0%*), TiNA on top of Cougar uses only 56% of the energy required by Cougar. For *tct=30%*, TiNA on top of Cougar only uses 24% of the energy required by Cougar, whereas the REM increases to 3.3%.

Figure 6 shows the energy savings and corresponding reduction in response time. The ratio of energy required compared to that needed by plain Cougar is on the X-axis, whereas the ratio of response time compared to that of Cougar is on the Y-axis. Because TiNA can send notifications instead of the readings for those cases that do not violate the *tct*, the time needed to hear from all children decreases. This decrease in response time is directly related to the savings in energy and will increase further with higher *tct* values. For the case of *tct=0%* (i.e., no relative error), the response time of TiNA on top of Cougar was 60% of Cougar's (while consuming only 56% of the energy required by Cougar). For the case of *tct=30%*, where only 24% of Cougar's energy was used, the response time when using TiNA on top of Cougar is 27% of Cougar's. This shows that using TiNA can result in substantial savings in energy and further decrease the query response time when used on top of Cougar.

Figure 7 shows the effects of TiNA on relative error and energy when used on top of TAG. In this case, TiNA on top of TAG uses between 86% (for *tct=0%*) and 74.9% (for *tct=30%*) of the energy of the TAG base case with corresponding REM of 0% and 3.7%. Compared to Cougar, these reductions are smaller. The reason behind this is that TiNA, when used on top of Cougar, saves energy both by not transmitting readings and by reducing the time spent listening. In TAG, however, TiNA only saves in transmission power since it must listen for its entire assigned time slice. This means TiNA uses the same receiving energy as in TAG, hence the savings for TiNA on

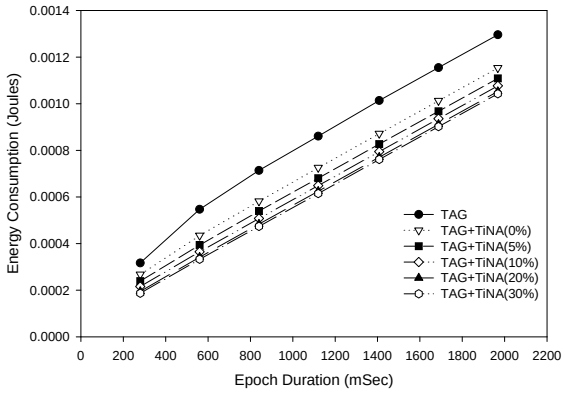

Fig. 8 Energy usage vs epoch duration

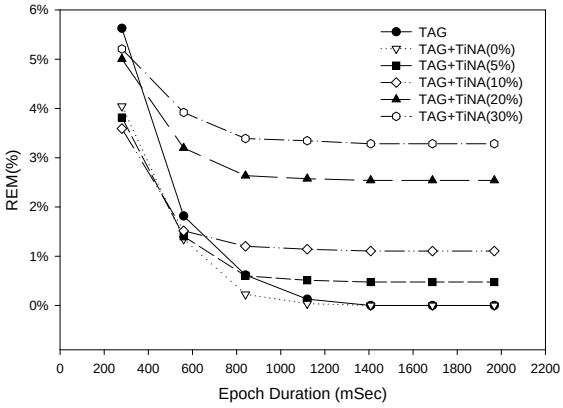

Fig. 9 REM vs epoch duration

top of TAG are limited to only transmission energy. Finally, we do not show the response times for TAG or TiNA over TAG, since they are always the same as the epoch duration.

In our experiments, we have focused only on the energy consumed for communication. In order to put these energy saving due to communication in perspective of the total system savings (as mentioned in Section 5) we need to take into consideration the cost of performing many other actions including sampling, processing, even dozing. These overheads depend on the type of the individual sensor nodes. Take for example the case of energy used for taking different kinds of readings (e.g., light, temperature, etc.) as reported in [22]. In this case, the savings provided by our scheme would depend on the kind of sensing activity. Sensing the temperature would require 90μJoules of energy while the transmission, as we reported above in the case of Cougar, would take 500μJoules for the regular implementation of Cougar and 280μJoules for Cougar with TiNA and *tct=0*. This leads to savings of 44%, however, including 90μJoules overhead for taking the reading, the savings drop slightly, to 37%. If instead of taking temperature readings a more costly sampling type was performed, for example, using a magnetometer which uses 1500μJoules to do a reading, the savings would only be 11%.

6.2 Sensitivity to the epoch duration

In cases where a sensor is in a dense portion of the network, its data will suffer high latency due to congestion and retransmission. Recall that in TAG, the epoch is broken up into communication slots during which a given level is sending and another level is listening. If the epoch duration is short (or if the routing tree is skewed) it may happen that the communication slot is too short for all nodes at a given level to successfully transmit their readings. In this case some nodes will be able to send their data while other nodes will not be able to transmit. This will give query results of poor data quality, as the aggregation will be missing data from sensors that failed to communicate their readings.

In this experiment, we test the effect of epoch size on TAG and compare it with TiNA on top of TAG. Up to now the epoch has always been large enough to allow all sensors to send in all their values during the assigned time slice. The goal of this experiment is to show to what degree TiNA can alleviate this problem by suppressing the transmission of relatively static readings in favor of the more dynamic ones which contribute the most to the quality of the final aggregate.

Figure 8 shows that by increasing the epoch duration, the amount of energy also increases in direct proportion. The reason for this is that as the epoch length increases, more sensors are able to send their readings. This behavior continues up to the point where all sensors can send all of their readings, after which the increase in energy is due to parent nodes having to listen for the entire time slice. The power used by TiNA follows a similar pattern. The amount of savings by TiNA decreases as the epoch time increases, since all the energy savings by TiNA on top of TAG are on transmission power.

Figure 9 shows the effects on REM when we increase the epoch size. In this case, we only get 0% relative error if there is enough time for all sensors to send their data. This figure shows that for smaller epochs, TiNA on top of TAG has a better QoD than TAG. The reason is that TiNA is ”wisely” choosing to send only those values that change, allowing parents to have more information with which to produce more accurate results. While TAG also has a cache to use, it is sometimes sending data already in the cache which means some new information cannot be sent. After a certain point, the epoch size is large enough to accommodate all nodes being able to send, and thus the relative error levels off for all schemes. Clearly, TAG will be dependent on finding the optimal epoch size for each workload, so we decided to use Cougar for the remaining experiments.

6.3 Sensitivity to degree of in-network aggregation

In this experiment we show the behavior of TiNA under different degrees of in-network aggregation. The degree of in-network aggregation is controlled by setting the number of attributes in the Group-By clause.

Figure 10 shows the energy usage for Cougar and for TiNA with various *tct* levels in the cases when we have no Group-By, when we have Group-By with one attribute, and

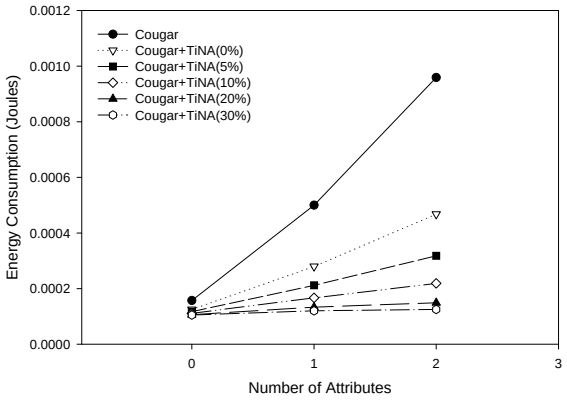

Fig. 10 Energy vs number of attributes

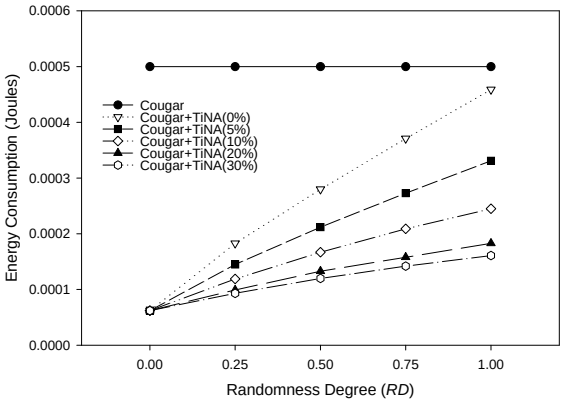

Fig. 11 Energy vs Randomness Degree

when we have Group-By with two attributes. In-network aggregation is at the maximum in the case of no Group-By where all sensors belong to the same group and a parent is always able to aggregate its reading with the readings reported from its children. On the contrary, in-network aggregation cannot be used in the case of a Group-By query with two attributes (i.e., the X and Y coordinates). In this case, each sensor represents its own group and thus a parent and a child can never belong to the same group. Hence, the performance of the network is similar to that provided by centralized query processing.

Figure 10 shows that energy usage increases when the number of attributes increases. However the rate of increase in energy usage for TiNA is less than that for Cougar. For instance, consider the performance of TiNA(0%) in the case of no Group-By attributes and in the case of 2 Group-By attributes. In the case of no Group-By (i.e., maximal in-network aggregation), using TiNA on top of Cougar reduces the energy consumption by 20% compared to plain Cougar; this reduction is 52% for Group-By with 2 attributes (i.e., no in-network aggregation).

6.4 Sensitivity to variability of sensor readings

This experiment focuses on the rate at which data changes and on the magnitude of change. We compared Cougar against TiNA on top of Cougar with various *tct* levels. Figure 11 shows the energy savings based on the rate of data change. The behavior of Cougar is not affected by the rate of data change (top line in graph). In the case of RD=0, TiNA uses almost no power, since it exploits the fact that sensor readings do not change. Actually, each node only sends once throughout its life; the rest of the time it only sends very small notification messages. At a 50% change rate, TiNA's power savings range from 44% (for *tct*=0%) to 76% (for *tct*=30%). In general, the less often data changes, the higher the chance that readings will be the same as before and the greater the chance to save on transmission costs when using TiNA. The worst case will be when data is completely random (RD=1), which is atypical for most applications. Even in this case, we show energy savings between 8.2% and 68%. The 8.2% energy savings for TiNA(0%) compared to Cougar occurs because TiNA exploits the few cases where although the individual sensor readings change, some of the partial aggregates remain the same (and thus the aggregate is not transmitted).

tct	0%	5%	10%	15%	20%	30%
REM	0%	0.5%	1.2%	1.7%	2.6%	3.3%

Table 2 REM vs. *tct* (RD=0.5) for TiNA over Cougar

Table 2 shows the relative error for each of the different tolerance levels. This table shows the case where the randomness degree is set at 0.5, the default case for our experiments. Even as the *tct* increases, the relative error increases at a much slower rate. For example, in the case where *tct* = 30%, the increase in relative error is only 3.3%.

Figure 12 shows the energy consumption based on the magnitude of change (i.e., Random Step Size Limit). The figure shows that Cougar is not affected by the magnitude of change (the same behavior shown in Figure 11). It also shows that, for a fixed *tct* value, increasing the step size results in increasing the energy consumption. This increase in energy is due to an increase in the rate of readings violating the *tct* requirement between successive rounds. This also explains the decrease in relative error when the step size increases, as shown in Figure 13. At a step size that is relatively small compared to *tct*, a reading could change many times while still being within the specified tolerance level, however, these changes contribute to an increase in relative error. Increasing the step size makes the change more noticeable and hence more changes are captured by the *tct* filter which leads to the shown decrease in REM, i.e. an improvement in QoD.

The above observation is further illustrated using Figure 14. In this figure we used the sinusoidal data model to generate readings as explained in Section 5. In this setting, all sensors generated readings that were *in phase* (same ϕ), however, we varied the sinusoidal period (T) between runs. In other words, a sensor reading is changing all the time and the magnitude of change depends on the period. That is, a short period (i.e., 20) is equivalent to a high frequency and a significant change in value between successive readings, while a longer period (i.e., 60) is equivalent to a low frequency sinu-

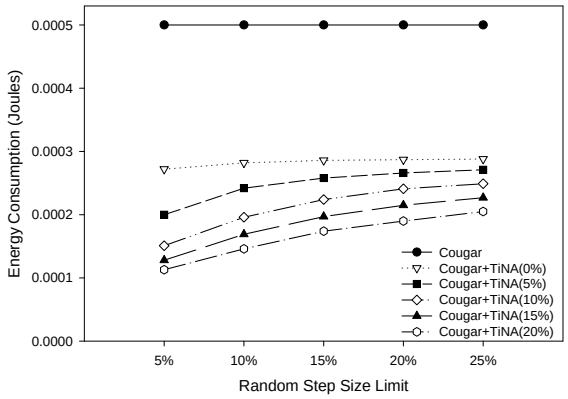

Fig. 12 Energy vs Step Size

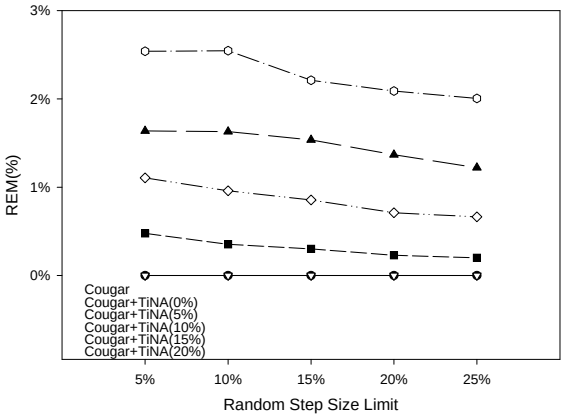

Fig. 13 REM vs Step Size

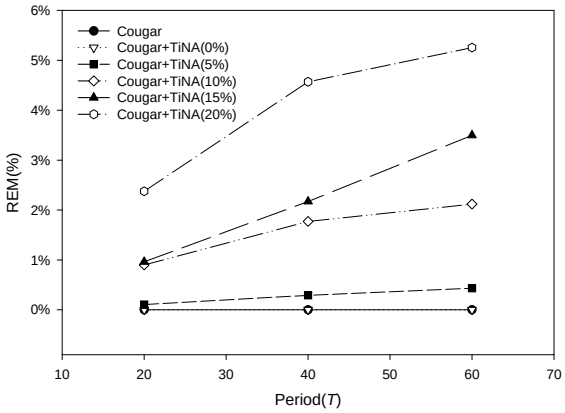

Fig. 14 REM vs. Sinusoidal period (T)

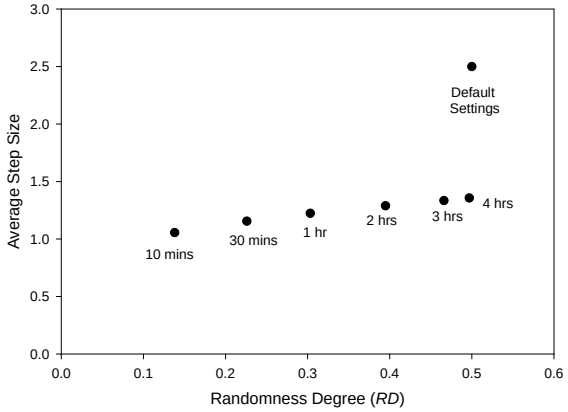

Fig. 15 Real Data Analysis

soidal and a small change between successive readings. The results presented in Figure 14 show how the REM decreases by decreasing the sinusoidal period, which agrees with the results previously shown in Figure 13.

Real Data Analysis: In order to put our results in a real perspective, we analyzed some available environmental readings from [25] which were previously used in [18]. We conducted this analysis to extract the real data variability parameters, namely, randomness degree and step size.

This data is gathered as part of the tropical atmosphere ocean project and the measurements include surface winds, sea surface temperature, upper ocean temperature and currents, air temperature, and relative humidity. Samples are taken at a resolution interval of 10 minutes and telemetered to shore in real-time via a satellite system.

In our analysis, we processed the air temperature readings for the year 2003 which was available from 15 stations. We converted the measured temperatures to the Fahrenheit scale and we assumed that a reading does not change if its integer part stays constant. We computed the variability parameters mentioned above given the default 10 min sampling rate. Moreover, we computed the same parameters when readings are sampled at longer intervals (up to 4 hours).

Figure 15 shows the randomness degree and average step size when readings are sampled at different rates, as well as our default settings. The figure shows that, in general, the randomness degree and average step size increase by increasing the sampling interval where the gaps are longer between successive readings. It also shows that our default experimental settings are realistic and more demanding than the real data settings. In our default settings, we set the randomness degree to 0.5 and the step size limit to 5 degrees (i.e., 2.5 degree on average). Compared to real data sampled at 10 minutes intervals, our experimental setting values are $\simeq 3.6$ times and $\simeq 2.3$ times the observed randomness degree and average step size respectively.

6.5 Alternative approximation method

In this experiment, we are comparing TiNA to an alternative approximation method, namely, sampling. Sampling can be used to reduce the number of exchanged messages and consequently the transmission energy. This can be achieved by excluding entire epochs; during an excluded epoch all sensors are idle and no result is reported by the network.

As in TiNA, sampling will provide approximated answers where there is an error between the reported aggregates and the exact ones. However, in the case of sampling this error is unlimited, while in TiNA the error is limited by the user specified tolerance. Moreover, in TiNA the suppression or trans-

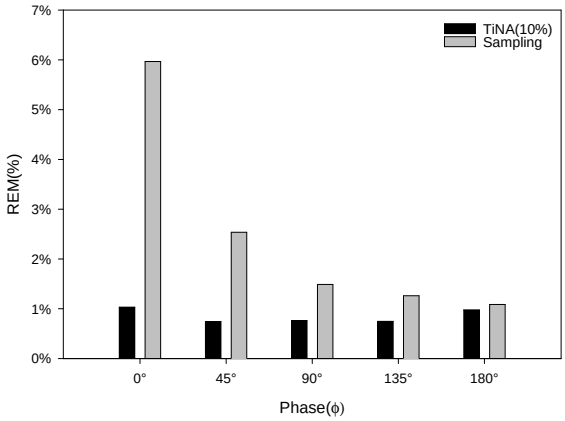

Fig. 16 REM vs. ϕ (tct=10%)

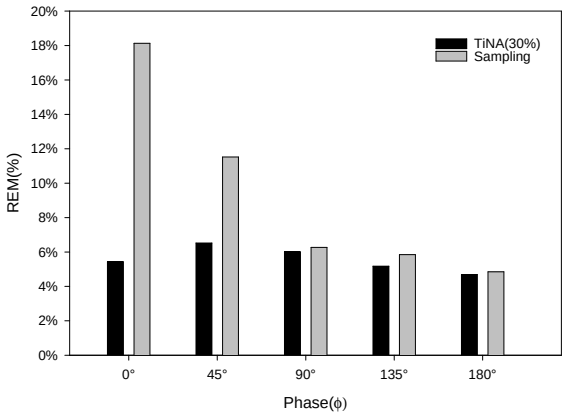

Fig. 17 REM vs. ϕ (tct=30%)

mission decision is guided by the previously reported state, while in sampling the decision does not consider the previous state.

In order to compare TiNA to sampling, we use the sinusoidal data model (which was presented in the previous section). The sinusoidal data model has the ability to control the temporal correlation between streams of readings generated by different sources, which is important in order to assess the quality of each approximation method. This can be illustrated using Figures 16 and 17. In both figures, the x-axis represents the maximum phase shift between any two sinusoidals generated by two different sensors. For example, in the case of $0°$, all sinusoidals are in phase, while in the case of $180°$ each sensor will use a sinusoidal where ϕ is randomly selected from the range $[0° - 180°]$.

Figures 16 and 17 show the REM provided by TiNA and by sampling under the same level of energy consumption. In order to make sure that they are both using the same energy, we measured the ratio between the energy used by TiNA to that used by Cougar (or TAG) and use this ratio as the effective sampling rate. Since this cannot provide uniform sampling intervals, we used that ratio to set the *probability of sampling*. Hence, using sampling, the probability of the network being active during an epoch is equal to that ratio. During the excluded epochs, reported results from the most recent epoch are assumed to be reported to the user and are used to measure the error.

Figure 16 compares the performance of TiNA(10%) with that of sampling. We can see that the average REM provided by TiNA is always less than that from sampling. However, the difference in error diminishes with the increase in ϕ. For example, consider the cases when $\phi = 0°$ and when $\phi = 180°$. In the former case, the error provided by TiNA is 5% less than that of sampling, while in the latter case, the resulting error is almost equal. The reason is that when $\phi = 0°$ all readings are either increasing or decreasing which leads to a continuous constructive interaction (as defined in Section 5), where the aggregated sum is always increasing or decreasing. Hence, exempting an epoch from sampling will increase the error significantly. Oppositely, when $\phi = 180°$, the readings generated by sensors at any point of time will span the complete sinusoidal, leading to an overall destructive interaction where the rate of change in the aggregate sum is low over time. This observation is further illustrated in Figure 17 where we compare TiNA(30%) to sampling. The figure shows that TiNA outperforms sampling by 13% in the case of $\phi = 0°$ and with only 0.2% when $\phi = 180°$.

Even though sampling provided a low overall *average* REM in the case of $\phi = 180°$, the provided individual relative errors per epoch show noticeable fluctuations. This is illustrated in Figure 18 where we plot the REM per epoch for TiNA(30%) and sampling at $\phi = 180°$. The figure shows that sampling provides a 0% REM when an epoch is included in sampling, while the error is unpredictable when the epoch is exempted. Moreover, it shows that the REM provided by TiNA will never exceed the specified tolerance (i.e., 30%). In the same experiment, we measured the maximum error over the simulation time and the standard deviation of the error. The results shows that TiNA(30%) exhibits a maximum error of 21%, with a deviation of 5%, while the sampling method shows a maximum error of 196%, with a 28% standard deviation.

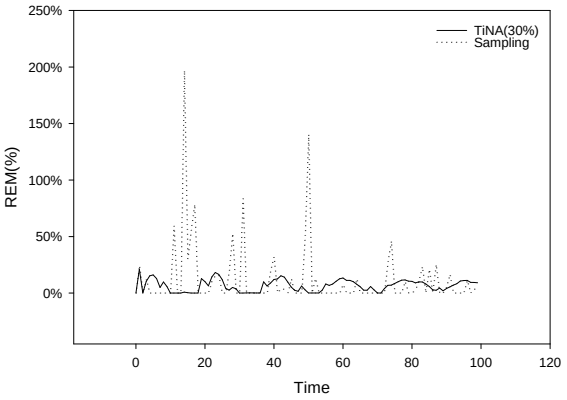

Fig. 18 REM vs. Time ($\phi = 180°$)

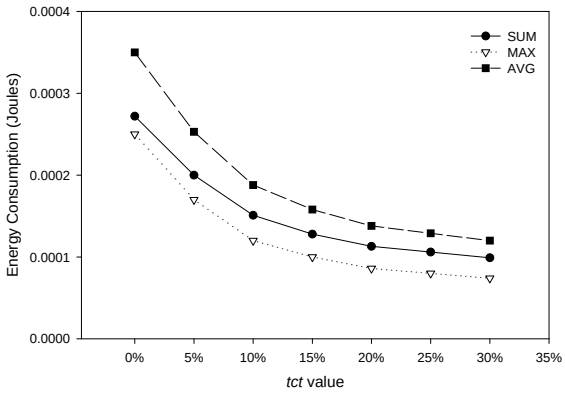

Fig. 19 Sensitivity to Aggregation Function (Energy vs tct)

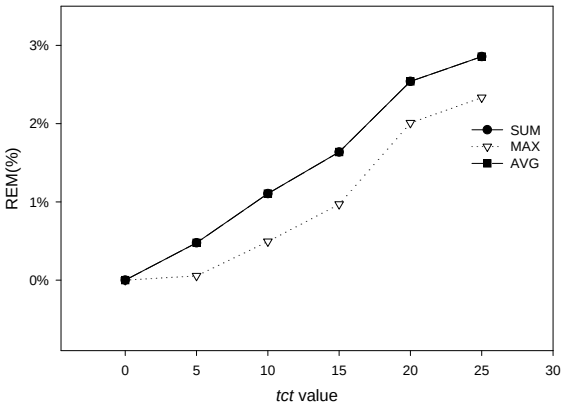

Fig. 20 Sensitivity to Aggregation Function (REM vs tct)

6.6 Sensitivity to the aggregation function

In this experiment, we test the sensitivity of TiNA and Cougar to the different aggregation functions. In Table 3, we report the energy used by Cougar and TiNA(0%). The numbers in parentheses represent the ratio of energy required compared to that needed by plain Cougar. The table shows that TiNA is using only about half the energy for each of the different aggregation functions. It also shows that for Cougar, both MAX and SUM use the same amount of energy, whereas in TiNA, MAX has lower energy requirements. This is because in MAX, changes at the reading level have less chance of affecting the previous partial aggregate value. Hence, one reported partial aggregate for MAX can stay valid for several rounds which leads to reduced transmission costs.

Aggregate	Energy for Cougar	Energy for TiNA(0%)
SUM	0.0005 (100%)	0.000272 (54.5%)
MAX	0.0005 (100%)	0.00025 (50%)
AVG	0.000654 (100%)	0.00035 (53.5%)

Table 3 Energy for TiNA(0%)

Figure 19 shows the energy consumption for TiNA on top of Cougar under different aggregation functions, with various *tct* levels. In this figure, we see that AVG has higher energy requirements than MAX and SUM. The reason for this is that for MAX and SUM, only the max/sum of the readings needs to be sent, while for AVG, the sum of the readings and the count is reported.

Figure 20 compares the REM under different levels of *tct* for various aggregation functions. We can observe that both AVG and SUM have the same relative error in all cases, while MAX provides a lower relative error. The former is because AVG is just the sum divided by the count, and the latter is because MAX as an aggregate is less sensitive to changes in the sensors readings as explained above.

6.7 Sensitivity to communication protocol overhead

One thing that can vary based on the wireless communication protocol is the size of the packet header. Because packet header size can affect the size of and number of packets needed to transfer data, we wanted to make sure that making the packet header larger or smaller would not dramatically affect our results. Note that changing the header size is equivalent to accounting for the additional control packets required by the MAC protocol (e.g., CTS and RTS).

In this experiment, we allowed the packet header to vary in size from 2 bytes to 8 bytes, while using 30 bytes as the size of packets. The results for this experiment are shown in Figure 21. All the results given are for the savings in energy normalized to the scheme on top of which TiNA lies. This means that TAG+TiNA(0%) would be TiNA with tct=0 normalized to the energy readings of TAG (i.e., Savings using *TAG+TiNA(0%)* = $\frac{|Energy_{TAG} - Energy_{TiNA(0\%)}|}{Energy_{TAG}}$), while Cougar+TiNA(0%) would be the same but normalized to the energy readings of Cougar.

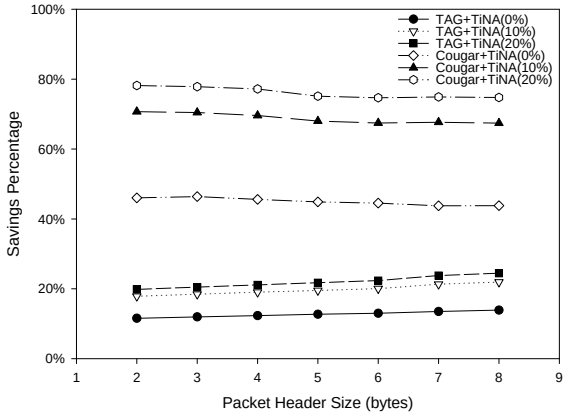

Fig. 21 Savings in energy versus base schemes for various packet header sizes

There are two observations from Figure 21. First is that as the packet header size gets larger, the savings from using TiNA on top of TAG actually increase. This is evident

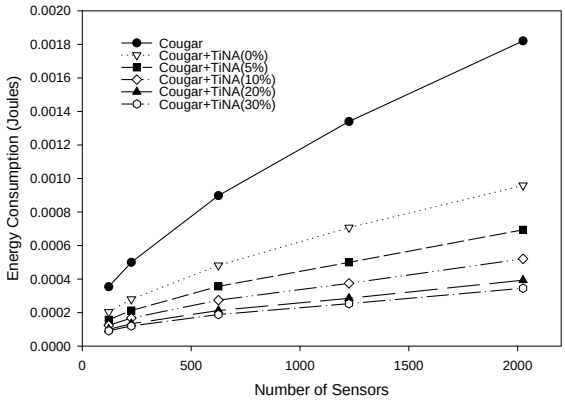

Fig. 22 Scalability: Energy vs number of sensors

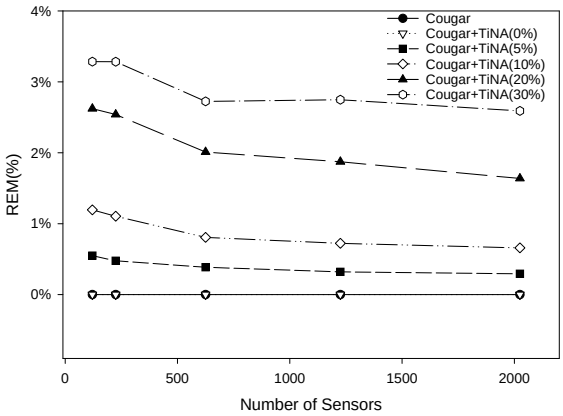

Fig. 23 Scalability: REM vs number of sensors

for the case of *tct=0*, where savings increased from 11.5% to 13.9% as the packet header size increased from 2 to 8 bytes. The reason for this is that as the packet header gets larger in size, the savings from sending no data (as will be done if the data does not change) also gets larger. In addition, there are cases where more packets need to be sent out as the packet header gets larger in size. Since there are only 30 bytes in each packet, if the packet header size is 8 then only 22 bytes exist for the packet payload. If the data is 28 bytes, this will require 2 packets to be sent in addition to the 8 bytes of overhead. With TiNA, these 28 bytes may be cut down to 22 bytes based on data staying within the *tct*, which allows only one packet to be sent and saves not only the 6 bytes of data, but also the 8 bytes of overhead.

The second observation is that the energy savings from Cougar decrease, but not by much. For example, in the case of a 2 byte packet header size and *tct=10* the energy savings were 70.7%, while with an 8 byte packet header size these savings decrease to 67.4%. The reason for this has to do with the notification packet used in TiNA on top of Cougar. Remember that the notification is only one bit, but the packet includes the header as well. With all else being equal, a bigger packet header will cause this notification packet to be larger, and hence cause more energy to be used. This will also cause nodes to listen for a slightly longer period of time because more bytes are being transmitted. The result is the decrease in energy savings, however this decrease pales in comparison to the savings of using TiNA on top of Cougar, showing that the packet header size does not dramatically effect the energy savings gained from using TiNA.

6.8 Scalability

In order to test the scalability of the proposed TiNA framework, we ran an experiment with increasing number of sensors in the network (i.e. increasing the grid size). Figure 22 shows the effect on energy when the size of the network changes: as the size of the network increases, the amount of energy saved also increases. In the case of the 11x11 network, TiNA is showing energy savings between 43% (for *tct*=0%) and 74% (for *tct*=30%). However, for the largest grid size we tested (45x45), we had savings of 48% and 81% for the same *tct* levels.

Figure 23 shows the REM for the various *tct* levels when the size of the network increases. The relative error actually decreases as the network increases in size. For example, for *tct*=30%, the relative error decreases from 3.3% when the grid size is 11x11 to 2.6% when the grid size is 45x45. The reason for this decrease is that in TiNA, a parent will send its reading if either the reading changes or the group it is a part of is already being sent up. As the grid size increases, there is a better chance the parent's group is being sent up already, and that the parent can therefore add its own reading in. This means more actual readings are being sent up the routine tree, thus decreasing the relative error, which results in an improved QoD.

6.9 Group-Aware Network Configuration

In this set of experiments we study the behavior of the proposed group-aware network configuration algorithm (GaNC) and also examine the synergy between TiNA and GaNC. We present results of using TiNA on top of Cougar and use the FHF network configuration method as our base case. We will denote the experiments where GaNC was used by adding +GaNC to the method.

For this set of experiments, the group IDs for the different nodes are assigned at random with values between 1 and N, where N is the total number of groups in a particular experiment.

6.9.1 Effects of Group-Aware Network Configuration

In this experiment we compare FHF with GaNC for varying *tct* amounts. We will show the effects of GaNC for network sizes of 15x15 and 45x45 and report both the energy savings and the quality of data. Figures 24 and 25 show the energy and relative error for the 15x15 size network and Figures 26 and 27 show the same for the 45x45 size network. The number of groups used in this experiment was set at 10.

The first observation is that for the most part, using GaNC decreases the amount of energy used by the sensor network.

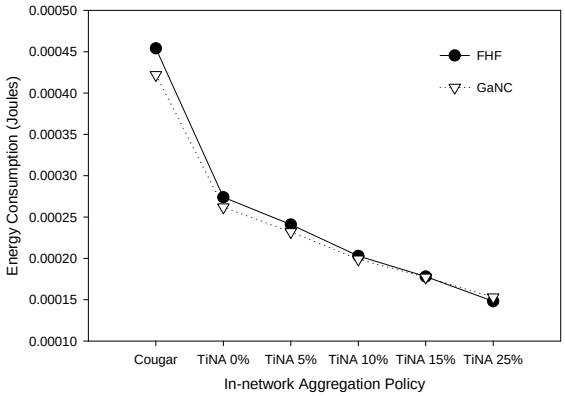

Fig. 24 GaNC in 15x15 Grid (Energy)

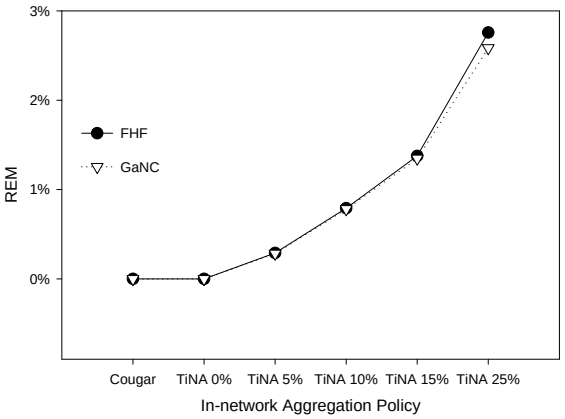

Fig. 25 GaNC in 15x15 Grid (REM)

This is especially prominent in larger networks. For TiNA with *tct=0%*, energy savings are 4.6% for the 15x15 grid and 29.3% for the 45x45 grid. The savings when using GaNC with plain Cougar are even higher: 7.1% for the 15x15 grid and 33.3% for the 45x45 grid.

This shows that the proposed group-aware network configuration method can reduce energy significantly even when it is not used in conjunction with the TiNA framework.

The next observation is that regardless of the size of the network, the energy savings of GaNC over FHF decrease as the *tct* increases. In fact, for the 15x15 network, there is cross-over point where FHF requires slightly less energy than the GaNC method, for high values of *tct*. This is illustrated in Figure 24 where GaNC saves 3.8% when *tct=5%* but uses 3.3% more energy when *tct=25%*. This result is not surprising: because of using GaNC, some nodes will switch to parents that are in the same group as themselves. While this tends to decrease the number of message a parent sends, there are cases where switching parents can cause more messages to be sent.

For example, assume that, using FHF, two children of the same group are routed through a parent of a different group. This would result in 3 messages being sent overall (one from each child and one from the parent further up the tree). If, however, when GaNC is used, each of these children change to be with a parent of their same group, they may end up choosing two different parents, because the parents in their same group are not close enough to be "clustered" together. In order to propagate information up the tree under this setup, 4 messages are needed (one from each child and one from each parent. This is a 25% increase in the total number of messages, which in turn causes an increase on the energy used in the network.

For larger networks, the positive effects of using GaNC will outweigh the negative effects (per our previous example). As the *tct* increases, there are less nodes that are transmitting, since their value changes are not violating the specified *tct*. In larger networks, since there are many nodes, there will still be a lot of nodes transmitting, even under high *tcts*. In Figure 26, we can see that for the 45x45 grid, the energy savings at *tct=0%* are 29.3% and they drop to down to 24.7% for *tct=25%* (howerver, there is no cross-over point in this case).

The final observation is that there is very little difference in the relative error between using GaNC versus using FHF to create the routing tree, even with the large savings in energy (from GaNC). For the 15x15 grid, the relative error has decreased in the case of GaNC. This decrease is minor, ranging from .005% for *tct=5%* to .2% for *tct=25%*, but exists nonetheless. This improvement is due to the "free" ride some

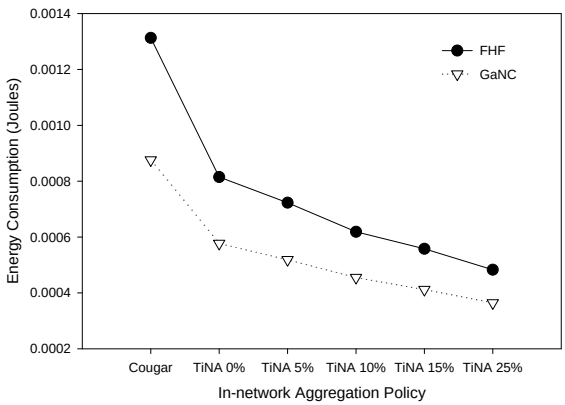

Fig. 26 GaNC in 45x45 Grid (Energy)

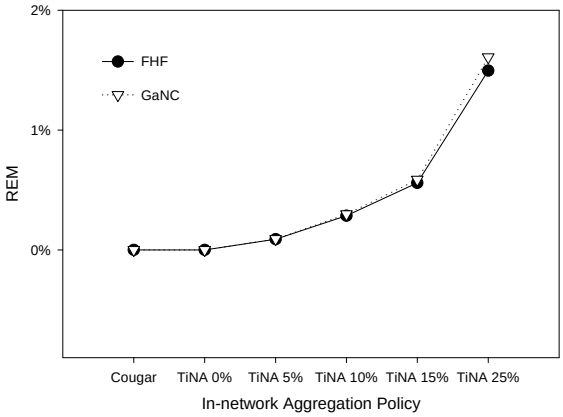

Fig. 27 GaNC in 45x45 Grid (REM)

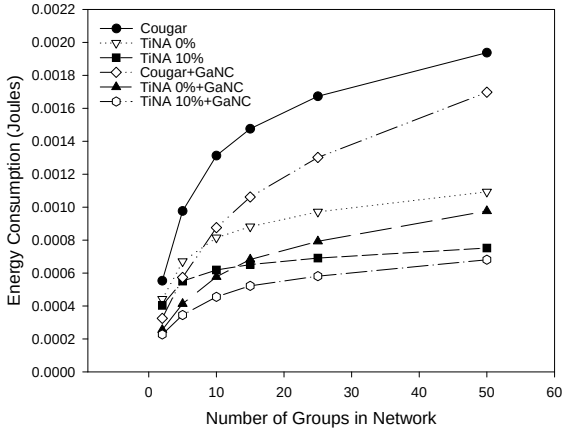

Fig. 28 Energy Comparison for varying number of groups

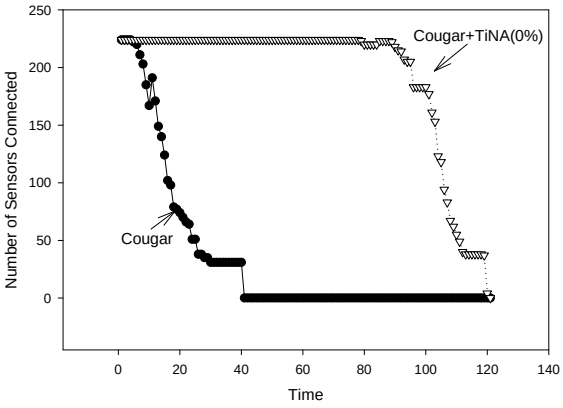

Fig. 29 Measurement of lifetime of the network

parent nodes may get by having a child already sending the same group as the parent and therefore be able to aggregate its own reading into the group aggregate without adding to the amount of energy used. In the case of the 45x45 grid, the relative error increases by a small amount. This increase was between .0001% for *tct=5%* and .1% for *tct=25%*. This small increase is explained by multiple nodes counteracting each others changes; thus the internal node does not get the ”free” ride it may have gotten with only one child changing values.

6.9.2 Synergy of TiNA and GaNC under varying number of groups

In this experiment we examine how the number of groups affects the behavior of GaNC. Figure 28 shows the results from this experiment. We compared plain Cougar, TiNA over Cougar with *tct=0%*, and TiNA over Cougar with *tct=10%*. We run two sets of experiments, one where FHF was used for configuring the network and one where we used GaNC instead (we note such cases with +GaNC). The number of groups ranged from 2 to 50. We had a total of 45x45 = 2025 nodes. With just 5 groups, GaNC is expected to reduce energy consumption significantly, since children nodes will be able to select parents that are in the same group as them (in other words, GaNC will have a lot of options to choose from). For all three cases, using GaNC instead of FHF saves 41% in energy for Cougar, 38% for TiNA(0%), and 37% for TiNA(10%).

Using GaNC when the number of groups is 50 will not reduce energy as dramatically as with the case of 5 groups. When the number of groups is high, there are less chances that a child can find a parent in the same group. Based on the grid configuration (which we used in our simulations), each child has a maximum of three different parents to choose from. With 50 different groups, the chance that the child is in the same group as one of those three parents is less than 1%, so the savings will be minimal.

This can be observed from Figure 28: for 50 groups the savings with GaNC are 12% for Cougar, 10% for TiNA(0%), and 9% for TiNA(10%). Overall, we see that when the number of groups increases, the savings with GaNC also decrease, but are still significant even when there are 50 different groups and the chance of a node actually switching parents in an efficient way is less than 1%.

6.10 Coverage

In the last set of experiments, we compared the network coverage provided by TiNA on top of Cougar versus that provided by plain Cougar. We define the *coverage* of the network for time t_i as the number of sensors that are still able to send their readings to the root sensor at that time. Obviously, sensors with depleted energy are not counted. However, a sensor may be still be “alive”, but if its parent is dead it is no longer counted for the coverage measurement. For that reason, we rebuild the routing tree for the sensor network periodically (every 5 intervals in this experiment). When the routing tree is reconstructed, a child chooses its parent in the same way as when the tree was first created. Since previous parent nodes may be dead, choosing former siblings as parents is possible. Finally, in our experiment, each sensor is limited to sending 5000 bits of information before dying.

Figure 29 shows our results for this experiment. We used TiNA with *tct* = 0%, which should maintain the same relative error as Cougar. Clearly, TiNA on top of Cougar outperforms the plain Cougar scheme, allowing sensors to stay alive significantly longer. If we define the *overall coverage* as the integral of coverage over the entire observation period (i.e. the area under each curve in Figure 29), then using TiNA can increase the overall network coverage by 270% compared to the plain Cougar approach. Also shown in this figure are several upwards bumps in the otherwise monotonically decreasing network coverage curves. A bump indicates that reconfiguration occurred, and several nodes that were previously disconnected because their parents had failed have now been given new parents which are still alive and thus these nodes are once again connected to the network.

7 Related Work

The idea of using different criteria to effect the building of routing trees has been examined on several fronts. One crite-

rion could be to use link quality to decide on the best parent to use because better link quality means less lost packets. However, determining the quality of the link[34] is not a trivial task. Another criteria is to use application semantics to do the data routing in sensor networks. This has been presented in [37,20], where the goal is to use information-directed routing in order to minimize communication cost while maximizing information aggregation. The work in [20] showed the significant gains of applying information-directed routing in locating and tracking moving objects.

Using the query semantics for efficient data routing was introduced in [22], which proposed the use of a semantic routing tree (SRT). The basic idea that motivates the use of SRT is the fact that a given query does not apply to all nodes in the network. Hence, those nodes for which the query does not apply can be excluded from the query in order to save communication costs. As in GaNC, the work on SRT considered optimizing the routing tree for the special case of constant-valued attributes (e.g., location). However, the objective of the SRT is providing a design to minimize the number of nodes participating in a query with a predicate over that constant-valued attribute. Whereas in GaNC, the objective is to cluster along the same path sensor nodes that belong to the same group in order to maximize the benefit of in-network aggregation in reducing the size of messages.

Providing approximate answers to queries is an approach that was first used to provide fast results when an exact answer is not essential or when producing the exact answer is very time consuming. Sampling is one technique that was used in [11,1] to provide such approximate answers over massive stored data. Using sampling, a posed query will be answered using summary statistics which allow producing an approximate answer and provide a fast response time. These summaries may either be generated online after the query is posed [11], or may be precomputed as in [1]. Moreover, the approximate answers are often supplemented with a statistical error bound to indicate the quality of the approximation to the user.

In networked database system, approximation remains as a very attractive technique to improve the utilization of the limited resources. The work in [7] demonstrates the use of wavelet-based summarization for storing data within the sensor network for future querying. The importance of approximation in networked database systems increases when it is required to continuously process continuously generated data. In that context, exploiting end-user tolerance to temporal coherency has been used to reduce communications cost for dynamic data generated by Web servers, data streams, and sensor networks.

One approach for dynamic data reduction is to use numeric bounds and queries with explicit precision constraints. This is the approach that TiNA follows and which does not require a prior knowledge of the generated data pattern and it allows the user to control the desired degree of precision. The former feature is specially important for cases where the changes are abrupt and unpredictable, which makes this approach attractive for a wide range of applications.

Besides TiNA, this approach has been used in [26,5,30, 24]. In [5] the user specifies a temporal coherency requirement for data items on a Web proxy. The work shows that combining this coherency requirement with intelligent data dissemination techniques achieves efficient and scalable utilization of Web servers and network resources. A following work [30] focuses on selecting topologies and policies to reduce the number of messages in a network of repositories while taking into account the data and coherency needs of users attached to each repository.

In the context of data streams, the work in [26] proposes a technique for reducing the communication overhead resulting from rapid update streams. In this technique, users register queries with precision requirements at a central processor, which installs filters on remote data sources. These filters are dynamically adjusted to account for sources that change at different rates and magnitudes. However, this adaptive adjustment of filters is only feasible if data generated at different sources follow a certain pattern, which might not be the general case. In this paper, we are addressing the problem of message reduction without any presumptions about the data behavior. Moreover, this technique assumes a centralized system, which is equivalent to a one-level network. In a hierarchical sensor network, where in-network aggregation is typically used to support aggregate queries, additional procedures are needed to ensure the efficient interaction between message suppression and in-network aggregation. Such procedures were discussed along the steps describing the functionality of TiNA in Section 4.

The *PREMON* paradigm for motion detection uses the spatio-temporal correlation in sensor readings to reduce transmissions [8]. In PREMON, the base station monitors the readings of sensors and generates a prediction model for each sensor, and sends these models back to the sensors. The performance of the PREMON approach is highly dependent on the accuracy with which prediction models can be generated and the percentage of readings that can be successfully predicted by them. The *APTEEN* protocol [24], like TiNA, attempts to reduce the number of transmitted readings in a sensor network. APTEEN uses filters to exploit temporal correlation in order to suppress values (using a soft threshold). However, unlike TiNA, APTEEN does not consider the semantics of the queries (specifically of aggregation queries).

In the best of our knowledge, TiNA is the first framework that exploits temporal correlation to support energy-efficient, QoD-aware in-network aggregation. As explained above, TiNA allows the user to specify a temporal coherency tolerance which is satisfied by applying numeric bound filters on sensor readings and partial aggregates.

8 Conclusions

In this work we explored two ideas in order to further reduce energy consumption in the context of in-network aggregation: (1) *influencing the construction of the routing trees for sensor networks* with the goal of reducing the size of transmitted

data, and (2) imposing a *hierarchy of output filters* on the sensor network with the goal of both reducing the size of transmitted data as well as minimizing the number of transmitted messages needed to ensure a specified level of quality of data. Our proposed methods provide a balance between energy efficiency and quality of aggregate data in sensor networks.

Specifically, this paper makes the following contributions:

- Proposed GaNC, a new network configuration algorithm for sensor networks, which considers the semantics of Group-By queries and the properties of the sensor nodes.
- Proposed TiNA, a framework of hierarchical output transmission filters that can reduce energy consumption while bounding the loss in the quality of aggregate data, based on user preferences. Our framework is independent of the underlying synchronization protocol used for sending and receiving data between the sensor nodes.
- Presented extensive simulation results that show clear gains for each proposed method in isolation and bigger benefits when the two proposed schemes are combined.

We have shown experimentally that our schemes result in large savings in energy over typical in-network aggregation methods without significant loss in quality of data. Specifically, we showed that TiNA can reduce the energy used for communication by up to 60%. We have also shown that using GaNC in addition to TiNA can result in an additional 29% savings in energy.

We also experimentally compared TiNA to a temporal sampling method. For the same energy consumption levels, TiNA had up to 13% higher overall quality of data than sampling. However, the relative error when using sampling reached up to 196% in our experiment, while on the other hand, TiNA maintained the relative error within the specified bounds (30% in this case) at all times. Therefore, such temporal sampling cannot be used to reduce energy consumption when user-specified preferences must be satisfied.

TiNA has also been shown to increase quality of data in cases where the period to transmit is too small and can increase the lifetime of the network by 270% compared to existing approaches. The TiNA scheme is particularly attractive for exploratory applications. In this case, TiNA is tuned to minimize energy consumption during the preliminary analysis and identify interesting trends in the sensor network. Once a trend is identified and higher accuracy is needed, TiNA is tuned to increase the quality of delivered data while consuming minimal energy.

In conclusion, both proposed schemes (GaNC and TiNA) work synergistically with existing in-network aggregation methods to further reduce communication energy consumption while maintaining quality of data within user-specified preferences.

Acknowledgements We would like to thank the anonymous reviewers for their thoughtful and constructive comments.

References

1. S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proc of ACM SIGMOD*, 2000.
2. J. Beaver, M. A. Sharaf, A. Labrinidis, and P. K. Chrysanthis. Location-aware routing for data aggregation for sensor networks. In *Proc. of Geo Sensor Networks Workshop*, 2003.
3. P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of MDM*, 2001.
4. J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.
5. P. Deolasse, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic web data. In *Proc. of WWW10*, 2001.
6. D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.
7. D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution search and storage in resource-constrained sensor networks. In *SenSys*, 2003.
8. S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: Taking lessons from MPEG. *Computer Comm. Review*, 31(5), Oct. 2001.
9. J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building effi cient wireless sensor networks with low-level naming. In *Proc. of SOSP*, Oct 2001.
10. W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-effi cient communication protocol for wireless microsensor networks. In *HICSS*, Jan 2000.
11. J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *SIGMOD*, 1997.
12. J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *ISPN*, pages 63–79, March 2003.
13. J. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro.*, 22(6), 2002.
14. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proc of ASPLOS*, 2000.
15. C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Inpact of network density on data aggregation in wireless sensor networks. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, July 2002.
16. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of MOBICOM*, Aug 2000.
17. P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Energy-effi cient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *Proc. of ASPLOS'02*.
18. I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *ICDE*, 2003.
19. C. Lin, C. Federspiel, and D. Auslander. Multi-sensor single actuator control of hvac, 2002.
20. J. Liu, F. Zhao, and D. Petrovic. Information-directed routing in ad hoc sensor networks. In *2nd ACM international conference on Wireless sensor networks and applications*, 2003.
21. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of OSDI*, 2002.

22. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of ACM SIGMOD*, 2003.
23. A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of ACM WSNA'02*, 2002.
24. A. Manjeshwar and D. P. Agrawal. APTEEN: A hybrid protocol for effi cient routing and comprehensive information retrieval in wireless sensor networks. In *Proc. of IPDPS*, 2002.
25. M. J. McPhaden. Tropical atmoshphere ocean project, pacifi c marine environmental laboratory. http://www.pmel.noaa.gov/tao/.
26. C. Olston, J. Jiang, and J. Widom. Adaptive fi lters for continuous queries over distributed data streams. In *SIGMOD*, 2003.
27. C. Perkins. Ad-hoc on demand distance vector routing (AODV). Internet-draft, Nov. 1997.
28. S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, S. Shenker, L. Yin, and F. Yu. Data-centric storage in sensornets. In *Workshop on Hot Topics in Networks*, 2001.
29. H. Schwetman. CSIM user's guide. MCC Corp.
30. S. Shah, S. Dharmarajan, and K. Ramamritham. An effi cient and resilient approach to fi ltering and disseminating streaming data. In *VLDB*, 2003.
31. M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Tina: A scheme for temporal coherency-aware in-network aggregation. In *Proc. of MobiDE*, 2003.
32. D. B. Terry, D. Goldberg, D. Nichols, and B. M. Oki. Continuous queries over append-only databases. In *SIGMOD*, 1992.
33. A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *ACM Mobicom*, July 2001.
34. A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys*, November 2003.
35. Y. Yao and J. Gehrke. Query processing for sensor net. In *Proc. of CIDR*, 2003.
36. M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *MASCOTS*, 2002.
37. F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, vol. 19, 2002.