Power-Aware In-Network Query Processing for Sensor Data[*]

Jonathan Beaver, Mohamed A. Sharaf, Alexandros Labrinidis, Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
{beaver, msharaf, labrinid, panos}@cs.pitt.edu

Abstract

Minimizing energy consumption has been a major objective at all levels in sensor networks. In this paper, we present TiNA, an in-network aggregation scheme that maintains the user-specified quality of data requirement while significantly reducing the overall energy consumption. Specifically, since communication dominates power usage in sensor networks, TiNA exploits end-user temporal coherency tolerances to reduce the amount of information transmitted by individual nodes. Further, we show that TiNa, by using temporal coherency tolerances, can allow for better quality of data when the time given to perform readings is too short for all data to be propagated up through the network. We compare our proposed scheme against an existing in-network aggregation scheme with a local sensor cache. We present experimental results measuring both power savings and also the quality of data for group-by queries. These results show that TiNA can reduce power consumption by up to 60% without any loss in the quality of data and extend the life of the sensor network by up to 300%.

1 Introduction

From monitoring endangered species [8, 12], to monitoring structural integrity of bridges [9], to patrolling borders, sensor networks offer today an unprecedented level of interaction with the physical environment. Within a few years, miniaturized, networked sensors have the potential to be embedded in all consumer devices, in all vehicles, in the human body for medical purposes, or as part of continuous environmental monitoring.

Sensor nodes, such as the Berkeley MICA Mote[5] which gathers data such as light and temperature, are getting smaller, cheaper, and able to perform more complex operations, including having mini operating systems embedded in the sensor[6]. While these advances are improving the capabilities of sensor nodes, there are still many crucial problems with deploying sensor networks. Limited storage, limited network bandwidth, poor inter-node communication, limited computational ability, and limited power still persist. In this paper we focus on the problem of limited power in sensor nodes.

One way to help alleviate the problem of limited power is through in-network query processing rather than query processing at the base station. For example, assume a query which counts the number of sensors in the network. One way to implement this is to have each sensor count itself and send that count up the network to the base station, with intermediate nodes just routing packets up the network. Another way, with in-network query processing (or

[*]This is an extended version of our paper "TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation", which appears in the Proceedings of the 3rd ACM MobiDE Workshop (September 2003, held in conjuction with MobiCom 2003). This work is supported in part by NSF award ANI-0123705. The second author is supported in part by the Andrew Mellon Predoctoral Fellowship.

aggregation), would be for each node to send the count of itself and all of its children. In this way, only one packet needs to be sent per node and each intermediate node adds the count of itself to that of its children before sending information further up the network.

As the example shows, with in-network aggregation some 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 [21, 4, 7]. Therefore, being able to transmit less data (the result of the aggregation over having to forward all the packets) results in reduced energy consumption at the sensor nodes.

Our approach is to use *temporal coherency tolerances* in addition to in-network aggregation to save energy in the sensor network while retaining the user's specifi ed quality of data requirement [1]. Our scheme, which is called TiNA (**T**emporal coherency-aware **i**n-**N**etwork **A**ggregation) is independent of the underlying synchronization protocol used for sending and receiving data between the sensor nodes.

The basic idea underlying the temporal coherency tolerance is to send a reading from the sensor only if the reading differs from the last recorded reading by more than the stated tolerance. 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. Temporal coherency checks occur at the level of individual readings and are compared against the last reading available for a sensor node.

In order to specify temporal coherency tolerance, we introduce a new clause in the expression of continuous queries. While it is the WHERE clause that acts as a (input) result fi lter, this new clause acts as a (output) transmission fi lter. By being a transmission fi lter, 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. For example, in group-by queries the length of the messages sent by a node depends on the number of groups existing in the routing tree rooted at that node. By not transmitting at the leaf level, there are cases where a group is no longer showing up at an internal node. This can propagate up the tree and result in additional savings for the sensor network.

We have experimentally evaluated our proposed temporal coherency tolerance scheme using simulation. We have 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. Specifi cally, these results show that TiNA can reduce power consumption by up to 60% and extend the life of the sensor network by up to 300%. 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 an existing in-network aggregation method.

The remainder of this paper is organized as follows. In Section 2 we provide a general overview of sensor networks and present related work. Section 3 discusses in-network aggregation in detail and, in particular, how TiNA, our scheme for temporal in-network aggregation, fi ts into sensor networks with different synchronization methods. TiNA is described in detail in Section 4. Section 5 presents our simulation environment and the performance metrics. In the same section, we also provide the details of the assumed energy model in sensor networks. Our experiments and results can be found in Section 6. We conclude in Section 7.

2 Background

Although there have been many advances in sensor network applications and technology, sensors still suffer from the major problems of limited bandwidth and have energy constraints. Understanding the events, measures, and tasks required by certain applications has been shown to provide effi cient communication techniques for the sensor network. In-network processing is one such technique that allows for data reduction to happen as close as possible to where data is generated [3]. In sensor networks, data processing is much cheaper than data communication in terms of time and energy consumption. This observation is what motivated the directed diffusion data dissemination paradigm [7].

Directed diffusion is data-centric, where data generated by a sensor node is named by 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 now self-identifying, this enables activation of application-specifi c caching, aggregation, and collaborative signal processing inside the network, which is collectively called in-network processing. Ad-hoc routing protocols (e.g., AODV) 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 to seize a data packet it is forwarding on behalf of another node, do in-network processing on this packet if applicable, and forward the newly generated packet up the path to the requesting node.

The work on TinyDB [10, 11] and Cougar [20] mapped the directed diffusion concepts in database terms. These projects used an SQL-style query syntax to express a node request for data. They also abstracted the 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, etc.) or data generated by this node (e.g., temperature, light, etc).

TinyDB and Cougar emphasize the savings provided by using in-network aggregation, which is one type of in-network processing. Sensor applications are often interested in summarized and consolidated data rather than detailed data. Such aggregate data is used as continuous input for data mining and analysis tools. Using in-network aggregations allows incremental computing of partial aggregates through the different paths in a hierarchical network of sensors. This reduces the number of packets exchanged between neighboring nodes and hence decreases the network's energy consumption and extends the lifetime of individual sensor nodes.

In-network aggregation requires the synchronization between sensor nodes on a single path to the root. A sensor node needs to wait to receive the readings from other neighboring sensor nodes before sending its own reading in order to increase the effectiveness of aggregation. This coordination allows a sensor node to compute the aggregate of readings from different sensor nodes in addition to its own reading and transmit the resulting partial aggregate.

The problem of deciding how long to wait is treated differently in the systems mentioned above. Directed Diffusion does not synchronize between nodes; data is forwarded immediately upon reception. However, this still enables duplicate suppression as a special case of in-network aggregation. TinyDB and Cougar use mechanisms that delay transmitting a sensor reading with the hope of aggregating readings from other sensor nodes. This delay is constant in TinyDB, while Cougar adjusts the delay interval based on the network status. The details of these two mechanisms will be discussed in the next section.

3 In-Network Aggregation Model

In this paper, we will look at queries that originate at the base station which will then forward the queries 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.

For routing, we use the general directed diffusion paradigm. The purpose is to build a routing tree where each sensor node is assigned a *gradient* [3] or a *parent* [10]. A parent sensor node for a child sensor node c is the sensor node through which c will send a message that it wants to propagate up the network. To build the routing tree, each sensor node i is assigned a level L_i and a parent P_i. Initially for all the sensor nodes, $L_i = \infty$ and $P_i = 0$. The root node that initiates a query sets its level (L_{root}) value to 0. Sensor nodes exchange query messages to build the routing tree. The header of such a message contains two fi elds: Id_s and L_s. Id_s is an identifi er of the source node that is transmitting the message, while L_s is the level value (i.e., L_i) of that source node. Building the routing tree proceeds as follows:

1. The root sensor prepares a query message which includes the query specifi cation. The root sensor also sets the (L_s) value in the message to its level value (i.e., L_{root}). It then broadcasts this query message to the neighboring sensor nodes.

2. A sensor node i that receives a query message and has its level value currently equal to ∞ will set its level to the level of the sensor node it heard from, plus one. That is, $L_i = L_s + 1$.

3. Sensor node 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 sensor node i in the network receives a copy of the query message and is assigned a level L_i and a parent P_i.

Since queries in sensor networks are continuous, new results are generated periodically. Such queries can be expressed using an extended SQL select statement:

```
SELECT {attributes, aggregates}
FROM sensors
WHERE conditions
GROUP BY {attributes}
HAVING conditions
EPOCH DURATION i
VALUES WITHIN tct
```

The fi rst fi ve clauses are the same as in standard SQL. The new clause *EPOCH DURATION i*, which takes one parameter i, was introduced by TAG [10] (the aggregation service for TinyDB). Parameter i is the epoch interval and it specifi es 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. The clause VALUES WITHIN *tct* is an addition made by TiNA to express temporal coherency tolerance.

The way in-network aggregation is performed depends on the query *attributes* and *aggregates*. Specifi cally, the list of attributes in the group-by query 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 if they belong to the same group. The aggregate 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.

As we mentioned in Section 2, synchronizing the transmission between sensor nodes on a single path to the root is crucial for efficient in-network aggregation. A parent sensor node has to wait until it hears the readings reported from all its children before reporting its own reading. Thus, a parent sensor node p is able to combine the partial aggregates reported by its children with its own reading and can then send one message representing the partial aggregation of values sampled at the subtree rooted at p.

Synchronization in TAG is accomplished by making a parent sensor node wait for a certain interval of time before reporting its own reading. Specifically, TAG subdivides the epoch into shorter intervals, with the number of intervals equal to the maximum depth of the routing tree (d) with the duration of each interval as i / d, where i is the epoch duration. During one interval, a parent sensor node will be active receiving messages from its children. In the next interval, these children sensor nodes will be idle, while the parent is still active transmitting the partial aggregation. This parent sensor node will then become idle in the subsequent interval, after it has finished receiving and transmitting the partial aggregation in its subtree.

Cougar uses a different approach in solving the synchronization problem: a parent sensor node will keep a list of all its children, which is called the *waiting list*, and will not report its reading until it hears from all the sensor nodes on its waiting list. This will result in different intervals at different parts of the routing tree which depends on the communication density at a particular part in the tree.

Our proposed scheme will work with either synchronization method. With a synchronization scheme like TAG, no change needs to be made for TiNA to work. The reason is that there are predefined times to send and listen, so all a parent needs to do is wait until its listening time is up. In a synchronization scheme such as Cougar, children need to notify parents in order for the parent to use its waiting list to determine when it has heard from all of its children. To accomplish this, we have a child send a one bit *validate* message to its parent if the new reading does not violate the *tct* amount or if the *tct* is violated then the child sends the actual reading. In this way, a parent will hear from every child and no modification is needed to the Cougar scheme. In this paper, we decided to present TiNA using the TAG approach for the synchronization component in our system. The advantages of the TAG approach are the guaranteed delivery of a query result every epoch duration and the minimizing of the energy consumed by a sensor. The latter is because a sensor is required to be active for only two intervals during the epoch: 1) when it is receiving messages from its children, and 2) when it is transmitting the partial aggregation.

4 TiNA

In this section, we present TiNA, our proposed scheme to perform **T**emporal coherency-aware **in**-**N**etwork **A**ggregation. In sensor networks, analysts pose continuous queries over a massive number of sensors. Periodically transmitting all the sensor readings up the network requires a lot of energy and shortens the lifetime of the network. Moreover, if the query rate is very high or the communication is very dense, the network will not be able to provide exact answers within the specified duration.

In TiNA, we are exploiting the temporal correlation in a sequence of sensor readings to reduce the energy consumption and at times increase the quality of data. Other systems, such as PREMON [2] and APTEEN [13], have also looked at using the correlation of readings to suppress messages. The former uses prediction and the later proposes the use of application specific threshold filters to do suppression. In the best of our knowledge, TiNA is the first scheme that exploits the temporal correlation to support energy-efficient quality of data in the context of in-network aggregation.

Another feature of TiNA is that it can be tuned to further reductions in energy consumption while still providing high quality approximated answers. Since approximation in TiNA is guided by the temporal correlation, the decrease in the quality of data is marginal compared to the cases when approximation is random. The TiNA approach is particularly attractive for exploratory applications, where TiNA is tuned to minimize the energy consumption during the preliminary analysis to identify an interesting trend in the network. Once the trend is identifi ed and higher accuracy is needed, TiNA is tuned to increase the quality of delivered data while consuming minimal energy.

4.1 TiNA scheme overview

TiNA is built to be used by any system that is doing in-network aggregation to increase the savings in energy throughout the entire sensor network. In these in-network aggregation schemes, aggregation is done at the internal nodes as information is passed up the routing tree. The contribution of TiNA comes in how it decides what information to pass up the tree. Normally, readings are passed up the tree once per period as defi ned in the query of the network. As mentioned in Section 3, TiNA adds a new clause to the query specifi cation called VALUES WITHIN *tct*.

The new clause of VALUES WITHIN *tct* is used by the user to specify the queries temporal coherency tolerance. The temporal coherency tolerance *tct* of a given query is executed at the reading level. The value of *tct* specifi es the degree to which the user is tolerant of changes in the value of readings. An example value for *tct* would be 10%, in which case the user is stating to report changes in the readings only if the reading differs from the last reported reading by more than 10%. Values for *tct* can range from 0, which would stand for report readings if any change occurs, to any positive number. This is equivalent to saying $tct \in [0\%, \infty)$.

4.1.1 Handling the WHERE Clause in TiNA

As mentioned in Section 3, the WHERE clause is used in the same way as in standard SQL: the user specifi es selection conditions that must be met for the results to be returned. The question becomes how does the WHERE clause work in conjunction with TiNA, which also only reports when certain conditions are met, in this case when the readings violate the *tct*. In TiNA, the WHERE clause will be used fi rst to fi lter the data to determine what meets the conditions and needs to be reported. The *tct* will then be used on any readings that meet the WHERE conditions to decide if the value needs to be reported or not, based on whether it will add any signifi cant information to the result set.

The issue that arises when using both WHERE clauses and the VALUES WITHIN clause is how to determine the difference between a value not meeting the conditions and a value not being sent because it is inside the given tolerance range. To use TiNA with the WHERE clause, three situations need to be handled, which are shown in Figure 1.

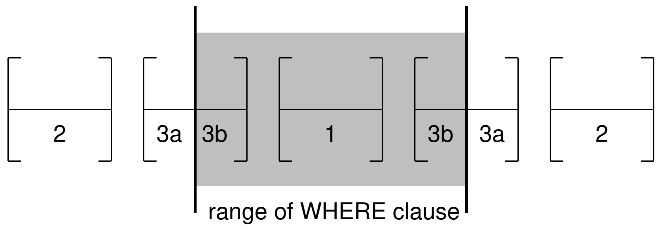

Figure 1: Cases Existing for TiNA with WHERE Clause

The situations marked 1 are situations where the range of the *tct* falls entirely within the area that meets the conditions of the WHERE clause (meaning the last reported value falls in the middle of the given area of the *tct*). In this situation TiNA behaves as it always would, sending results up the tree only if the value exceeds the tolerance. Otherwise, nothing would be sent from child to parent and the old value the parent has stored for the child is used. The situations marked 2 are those where the range of the *tct* falls completely outside the range of values that meet the WHERE condition. In this situation the *tct* does not have any effect on whether the data is sent or not. Regardless of whether the value exceeds the *tct* or not, the only time the data will be sent is if it meets the WHERE conditions, which is independent of the *tct*.

The fi nal situation is labeled 3 and is the situation where the *tct* falls on the border between meeting and not meeting the WHERE conditions. This situation has different actions that must be taken depending on what the last known reading is at the parent. If the last value fell in situation 3a (meaning the parent has an invalidated, empty value for the child), and the newest value falls in situation 3a, it is treated as in situation 2. If the last value fell in situation 3a, and the newest value falls in situation 3b, it is treated as it would be for a normal system and the value is sent because it now meets the WHERE condition that it did not meet before.

If the last value fell in situation 3b and the new value falls in 3a, an invalidatation message needs to be sent to the parent. The reason is that a parent in TiNA assumes on not receiving a value that the last value reported by a child should be used. In cases where the WHERE clause exists, a value can be invalidated, which tells a parent no value exists for a particular child. This means if an old value used to exist and a new one should not, the old value needs to be invalidated at the parent so it is no longer used. This invalidate message is just a single bit that lets the parent know to invalidate the stored reading for that child. The fi nal case is if the old value fell in situation 3b and the newest value is also in situation 3b. In this case nothing is sent because the new value does not violate the given *tct* and the old value can be used. For values that violate the *tct*, an invalidation is still sent if the value was valid and now is not. All other cases follow the rules mentioned above or the general rules of the WHERE clause.

In addition to the cases mentioned above, there is also a special case for the WHERE clause that affects how the *tct* is used. When the WHERE clause is with a condition that involves equal, any value for the *tct* is the same as having a *tct=0*. For example, imagine a query that is selecting light readings from a group of sensors. The WHERE clause in the query specifi es to only report those sensors whose light value is equal to 10. No matter what the *tct* is that is specifi ed by the user, the only way TiNA saves is if the value did not change since the last reporting. Since the WHERE is affected on the value fi rst, any *tct>0* will not occur because it has already been pruned out by the WHERE clause. Invalidation messages are also needed as mentioned above, but in this situation the invalidation is needed when a sensor used to meet the criterion and no longer does.

One fi nal issue that must be addressed is distinguishing between a value not being sent because it does not violate the *tct* and because the sensor node has died. To solve this problem, a child node that is suppressing transmission will periodically send a *heartbeat* message to its parent to let the parent know it is still alive. If a parent does not receive a heartbeat or any other message from a child after a certain period of time the parent will invalidate that child until it hears from the child again. This technique will also work for mobile sensors, where a sensor might change its location in the network, thus switching parents.

4.1.2 Features in TiNA

In TiNA, one of the most important features is keeping past information to use for comparisons. Depending on where in the network the sensor is (either a leaf or an internal node), the information kept is different. Leaf nodes keep only the last reported reading. The *last reported* reading is defi ned as the last reading successfully sent by a sensor to its parent. The reason we use this defi nition is to deal with cases where the reading slowly increases in value. If the last reported reading was defi ned as the last reading taken, small changes that eventually amount to a big change would not be identifi ed.

At the internal nodes, in addition to the last reported reading for that nodes, as explained above, the last view it received from each child is kept. This view can either be a single tuple (in the case of leaf nodes) or a partial view (in the case of other internal nodes). The way these past views are used in an important feature of TiNA. First, the parent receives information from its children. For each child, the parent compares the view it gets with the old view of that child and fi lls in any information that is missing. After the time to listen is over, the parent then gets the old view for every child it did not hear from and adds the old view of that child into the new partial view it is creating.

The internal node then takes its own reading. This reading is then added into the new partial view that is being created. The node at this point takes an old partial view (one made up of all old views and its own old reading) and compares it against the new partial view it created. For any tuples where the value has changed (just like using a *tct*=0 at the group level), that tuple is added to the result view that will be sent further up the tree. Old children views are then replaced if new ones were received and the result view is sent.

4.2 Example execution of TiNA

As an example, let us consider the performance of two systems, one with and one without TiNA. Figure 2 show the execution of a query during a single epoch in the fi rst system, that is, without TiNA, whereas Figure 3 in the second one, that is, with TiNA.

We assume the query involves getting the total light for rooms and grouping by floor, with the *tct* = 10%:

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

In the fi gures, nodes are represented as circles and the flow from child to parent is represented with arrows. The boxes connected to each node represent the current state at the node. This current state consists of the last reported reading, called Old, the current reading made, called New, and a table representing that node's view of the groups as it was before this round of readings. Tables along the connection lines represent the data that is being sent from child to parent. The Cost number under each of these tables is the cost to send this table from child to parent. The cost is really just the size of the table but is used to show 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. See Section 5 for the details on how total cost was computed.

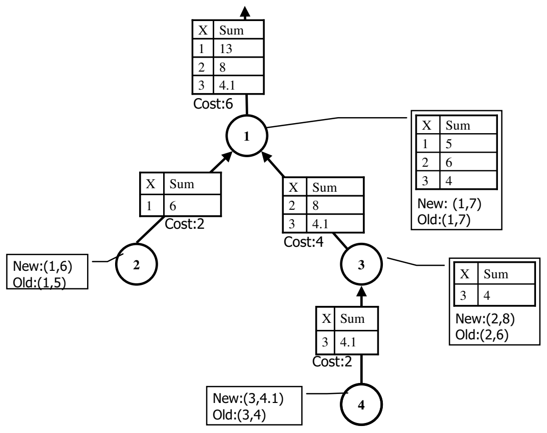

Figure 2: Epoch of query without TiNA

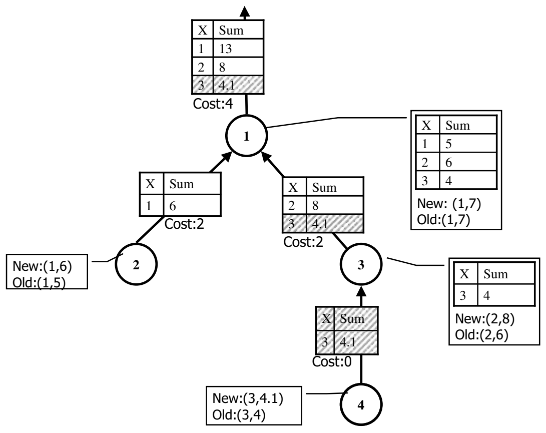

Figure 3: Epoch of query using TiNA

In Figure 2, we see a period of the query in a system that does not include the TiNA scheme. In this example, every reading is sent from child to parent[1] What we are concerned with is the total cost for the entire network. The

[1]Based on the adopted synchronization scheme, nodes at the same level in the fi gure send during the same time interval.

total cost for this network to send the readings up the tree is 14, or the sum of the sizes of all the messages is 14. As will be shown below, using TiNA will cut this total number of messages down and thus decrease the total energy used in the network.

In Figure 3, the benefi t of TiNA is shown. The set up is the same as above, but now the sensors network has employed into it the TiNA scheme. Savings from the TiNA scheme (parts of messages that do not need to be sent) are shown by being shaded out. As seen, when it is time for node 4 to send, it fi rst checks its new reading against its old reading. Since the new reading does not differ by more than 10%, nothing is sent to its parent. During the next time slice, nodes 2 and 3 check their readings against their last readings. Since both differ by more than 10%, they will both be sending their new readings and replacing their old ones. Node 1 then gathers all the readings it was sent. Node 1's reading does not differ by more then 10% from its old reading. However, the group it is a part of is already going to be sent further up the tree, thanks to Node 2. In this case, Node 1 will aggregate into Node 2's reading its new reading, as explained in Section 4.1. Node 1 then sends all of this new information up to the base station to be reported to the user. The total cost for this scheme is therefore 8.

As shown in the example execution of TiNA above, even in the case of four nodes, signifi cant energy can be saved. In the example, TiNA eliminated the entire transmission from Node 4. Due to the nature of sensor networks, being able to save on data transmission results in large overall savings. In addition, not having to send group 3 propagates up the tree. Both nodes 3 and 1 saved from Node 4 not changing by more than 10%. The size of their messages decreased and resulted in energy savings.

4.3 Discussion

As we showed in the previous example, TiNA can decrease the total size of transmissions by 43% for the overall network, compared to the method of sending every reading. This also shows the focus of the savings for TiNA. In TiNA, the focus is on cutting down on the number of transmissions that need to be made. Furthermore, TiNA focuses on cutting down message size for internal nodes. The reason for this focus is that doing the aggregations, taking the readings, and listening for data is the same in TiNA and TAG, and for most other methods of synchronization we looked at. Additionally, since the power usage of transmitting is much greater than that of processing, the small amount of additional processing needed to maintain the views is justifi ed when compared with the amount of power saved by not sending readings up the tree.

In addition to the obvious advantages of reducing the energy consumption, TiNA increases the quality of data in the cases of dense network traffi c or when a query's timely requirement is tight. In these cases, a sensor in a dense portion of the network will suffer high latency due to congestion and retransmission. If the epoch duration is not long enough (or if the routing tree is skewed), the resulting synchronization interval will be very short. Thus, some sensors will not be able to transmit their readings during this short interval. This will give an inaccurate query result, where the aggregation is missing tuples from sensors that failed to communicate their readings. Using TiNA alleviates 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 fi nal aggregate.

5 Evaluation Testbed

To study the effects of the proposed scheme, we created a simulation environment using CSIM [15]. The simulated network consisted of an 15x15 grid of sensor nodes; experiments with other grid sizes produced comparable results and are omitted due to space limitations. Each node could transmit data to sensors that were at most one hop away from it. In a grid this means it could only transmit to at most 8 other nodes. We simulated a contention-based MAC protocol (PAMAS) which avoids collision [19]. In this protocol, a sender node will perform a carrier sensing before initiating a transmission. If a node fails to get the medium, it goes to sleep and wakes up when the channel is free.

In the experiments included in this paper we compared the performance of our proposed scheme, TiNA, against TAG. For fairness in comparisons, we simulated the optimized version of TAG where a *child cache* is used [10]. 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, as proposed in [10], is particularly important in the cases where the communication is unreliable or the epoch duration is too short.

In the following experiments, the group-by query is of the form described in Sections 3 and 4, and the sensor network produces a result on every epoch duration. The objective of the query is to aggregate a measure (e.g., temperature) across different regions of the network. 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 an aggregate function. The SQL aggregate functions that we used in this work are: SUM, AVERAGE, and MAX. We did not include the MIN function, which is similar to MAX. We will focus on two measurements in the experiments: *energy consumption* and *quality of data*.

5.1 Energy Cost Model

In our experiments we assume the commonly used energy consumption model for wireless sensor networks [21, 4] in which there is a base station (BS) that can provide access to the network of sensors. The base station is assumed to have unlimited energy. All of the other sensor nodes in the network have limited energy and each sensor node consumes energy in four main activities: *transmitting, listening, processing,* and *sampling*. That is, the total energy consumption at each node is captured by:

$$T_{cost} = E_{transmitting} + E_{listening} + E_{processing} + E_{sampling}.$$

We focused on transmission power since the amount of time spent listening using the TAG synchronization is the same for all sensor nodes and sampling is dependent on the synchronization and tree building methods, of which our scheme is independent. We did not include energy required for processing because it is negligible compared to that needed for communication as was observed in [3].

The cost for transmission has to take into account not only the packet size *s*, but also the distance between the sender and receiver *d*. This makes the cost of the sender to be

$$E_{transmitting} = s * E_{Tx} + s * E_{Amp} * d^2$$

where E_{Tx} is the cost for using the transmitter (i.e., the bit cost for the transmitter electronics) and E_{Amp} for the amplifi er cost (i.e., the the per bit cost per square meter amplifi er cost).

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/constant amplifi er cost of transmitting data. In essence, this makes the sender cost above to only be $s * (E_{Tx} + E_{Amp})$. Therefore, the energy consumed in transmitting depends only on the aggregate size and the number of messages containing this aggregate.

In the experiments we report the average energy consumed by a sensor node per query period. The values of the parameters needed to calculate the transmission cost were the same as in [5]. Specifi cally, we simulated sensors

Parameter	Value	Default
Grid Size	15x15 Cells	
Aggregate	MAX, SUM, AVG	SUM
Number of Attributes	0, 1, 2	1
Epoch Durations	60 – 1000 mSec	1000 mSec
tct	0% – 25%	
Randomness Degree	0.0 – 1.0	0.5
Number of Epochs	100 Epochs	

Table 1: Simulation Parameters

operating at 3 Volts and capable of transmitting data at a rate of 40 Kbps at 0.012 Amp. transmit current draw. Hence, the energy cost of transmitting one bit in $Joules$ is computed as:
T_{cost} = 3 Volt * 0.012 Amp * 1/40,000 Sec = 0.9 μJoules.

5.2 Quality of Data

The other metric in our study is the *quality of data* (QoD). QoD is a measure of how close the exact answer and the approximate answer are. The exact answer is generated if all sensors deliver their current readings within the epoch time. An approximate answer is the 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 the change in the sample value is less than the *tct*.

In order to compute the QoD, we fi rst need to measure the error over the group-by query. We measured this error as described in [14]. 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 defi ned to be the relative error, i.e., $\epsilon_i = \frac{(|m_i - m_i'|)}{m_i} \times 100$. The error δ over the group-by query is defi ned as: $\delta = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i$ Finally, the QoD over time is defi ned as:

$$QoD = \frac{1}{T}\sum_{t=1}^{T} 100 - \delta_t$$

where δ_t is the query error at *epoch*$_t$.

5.3 Random Walk Model

Values for our experiments are generated following a random walk 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 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 sensed 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 10% limit implies that a new reading can differ by at most 10 (=10% of 100) compared to the previous reading. Simulation parameters are summarized in Table 1.

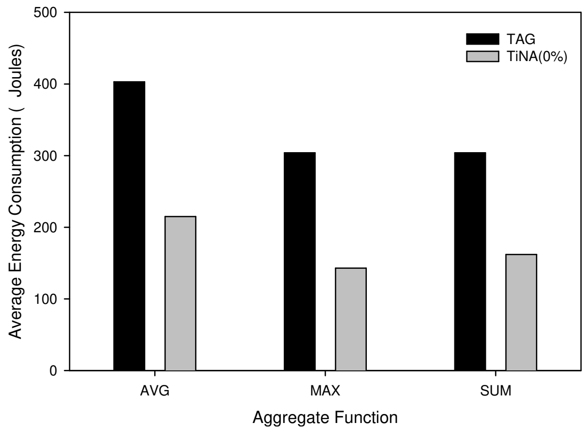

Figure 4: Energy for different aggregate functions

6 Experiments and Results

6.1 Sensitivity to the Aggregation Function

In the first experiment, we test the sensitivity of our scheme to the different aggregation types. Figure 4 shows the results of the base case (TAG) versus the case where *tct* is 0, which is the case where data is only sent if the current reading is different than the previously transmitted value. Our proposed scheme saves 46% for AVG and SUM and over 52% for MAX. With the degree of randomness at only 50%, half of the time the sensor readings do not change, which explains part of the savings. The additional savings come from reduced message sizes at the internal nodes.

From Figure 4, for both TiNA and TAG, 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 in AVG, the sum of the readings and the count is reported. For TAG, both MAX and SUM use the same amount of energy, whereas in TiNA, MAX has lower energy requirements. This can be explained by considering a simple scenario where several readings that belong to the same group change except for one. Assume that this unchanged value is the maximum value for that group in the previous epoch. In TiNA, all leaf nodes will report the changed values to their parent node regardless of the aggregate function (i.e. all but one). However, if the aggregate is MAX, the parent node will detect that the maximum value for the group did not change from the previous epoch and it will suppress transmission. This results in energy saving at this parent node and the nodes following it on the path to the root.

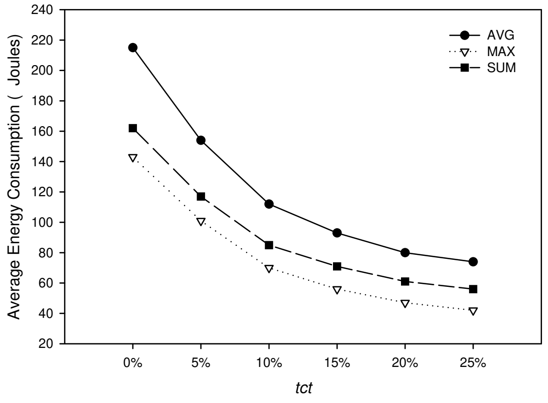

Figure 5: Energy for aggregate functions vs *tct*

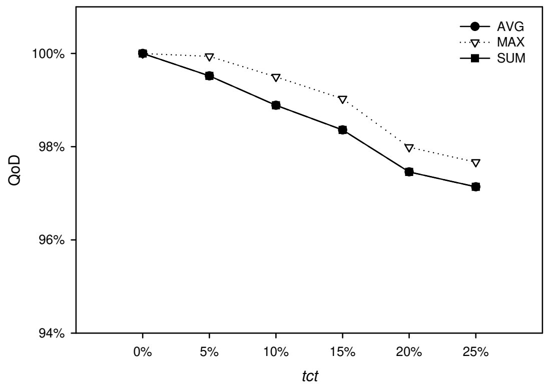

Figure 6: QoD for aggregate functions vs *tct*

Figure 5 compares the energy consumption for different *tct* for AVG, MAX, and SUM: energy steadily decreases as the *tct* value increases. The reason for this is that high *tct* values mean lower probability of a value change resulting in data transmission. Figure 6 shows the change in QoD for all the cases. First of all, for higher *tct* values, the QoD decreases. If some values are not being reported, then comparing the result against the case where every reading is reported would have discrepancies, hence the lower QoD. Secondly, both AVG and SUM have the same QoD in all cases, while MAX exhibits higher QoD. The former is because AVG is just the sum divided by the count, and the latter, we found to occur because in MAX, changes have a better chance of not affecting the overall MAX, but in other cases (i.e., AVG and SUM), any change affects the entire result.

6.2 Sensitivity to the Number of Attributes

In the next experiment we test how the number of attributes in the group-by clause affects the power consumption and QoD. Figure 7 shows the energy usage as the number of attributes is increased. As the number of attributes increases, the number of groups that are created by the group-by query also increases. As the number of groups increases, the savings from using TiNA also increase. In addition to showing the benefi t of TiNA for different amounts of groups, it also shows the scalability of TiNA. The more groups we have, the larger the savings because of the greater chance of savings at the leaf level (and thus decreased number of messages), and at the internal nodes (and thus decreased message sizes).

Figure 8 shows the trade-off between energy savings and QoD for the different number of attributes. We present the QoD and the energy savings of TiNA compared to TAG for *tct*=10% computed as $\frac{|E_{TAG}-E_{TiNA}|}{E_{TAG}}$. In this case, the QoD has only decreased by at most 2% for any of the dimensions, while the energy savings increase constantly. Even in the case where every node represents a distinct group (attributes=0), QoD has only decreased by 2% while at every leaf the value has to change by more than 10% to be used.

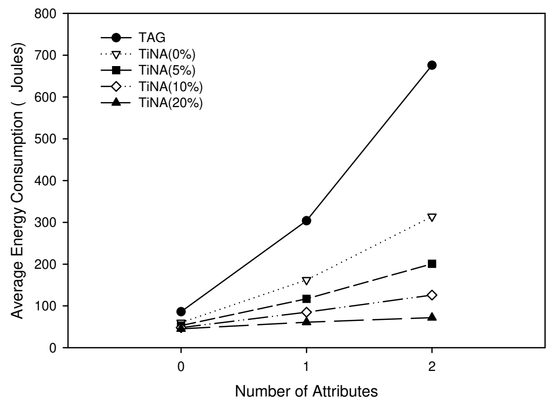

Figure 7: Energy vs number of attributes

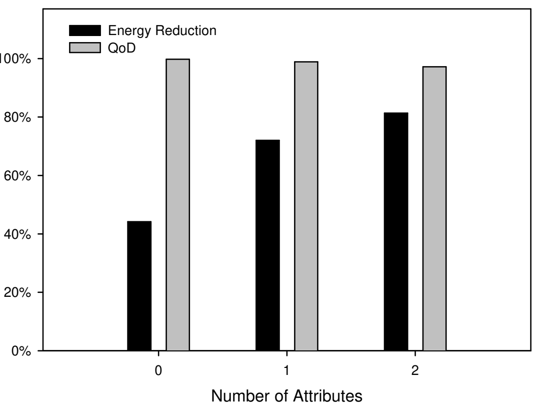

Figure 8: QoD and Energy reduction of TiNA(10%) compared to TAG for different number of attributes

6.3 Sensitivity to the Data Change Rate

The next experiment focuses on the rate at which data changes. Figure 9 shows the power savings based on the rate of data change. In the case of *RD*=0, TiNA uses almost no power. In fact, each node only sends once throughout its life. At a 50% change rate, our power savings are over 45%. For the case where data is completely random (rate of change equals 1), we show savings between 10% and 40%. Since most readings will have some redundancy, the case

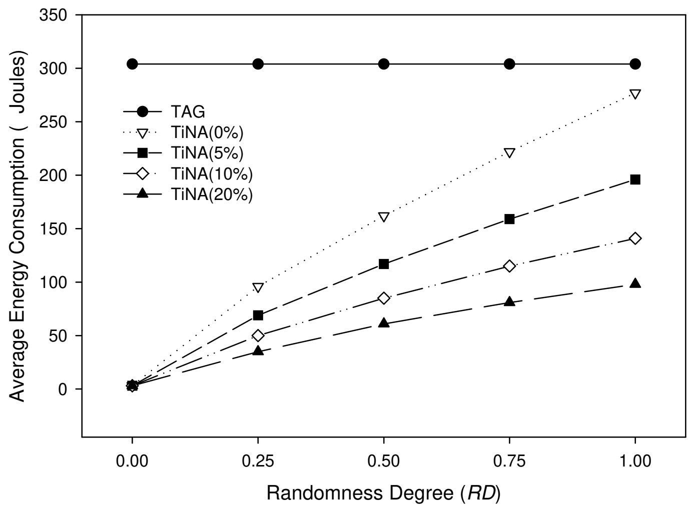

Figure 9: Energy consumption vs Randomness

where the rate of change is one is the worst case for our scheme. The explanation for power savings is that 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.

tct	0%	5%	10%	15%	20%	25%
QoD	100%	99.5%	98.8%	98.3%	97.4%	97.1%

Table 2: QoD vs. *tct* (*RD*=0.5)

Table 2 shows the quality of data for each of the different tolerances. This table shows the case where the randomness of data is set at 0.5, the average case for our experiments. Even as the *tct* increases, the QoD decreases at a slower rate. For the case where *tct* is set at 25%, we are only showing a 3% decrease in QoD. This again shows that the *tct* causes a small decrease in QoD when compared to the energy savings (80% in the case of *tct*=25%).

6.4 Temporal Coherency on Partial Aggregates

We ran another experiment using different *tct* values at the partial aggregates level in addition to the readings level as described in Section 4.1. Table 3 shows the results of that experiment using tct=10%. Compared to only applying *tct* on the readings level, applying *tct* at the partial aggregates level helps reduce the energy by about 12% with only a 2.5% decrease in quality of data However, we obtained the same results by applying *tct*=30% only on the readings level. Therefore, the behavior of applying the *tct* on the partial aggregate level can be also obtained by applying a higher-value *tct* on the readings level.

***tct* applied on:**	**QoD**	**Energy Reduction**
Sensor Readings	98.8%	72%
Partial Aggregates	96.3%	84%

Table 3: *tct* on readings vs. on partial aggregates

6.5 Effect of TiNA on Network Lifetime

In this experiment we looked at the effect of TiNA on the lifetime of the network. In this paper, we defined the lifetime of the network as the number of sensors that are still active (able to send) at any point in time. Each sensor is limited to sending 200 bits of information before dying[2]. The other parameters used are the defaults as mentioned in Table 1. The measurement is number of sensors alive during a given query period.

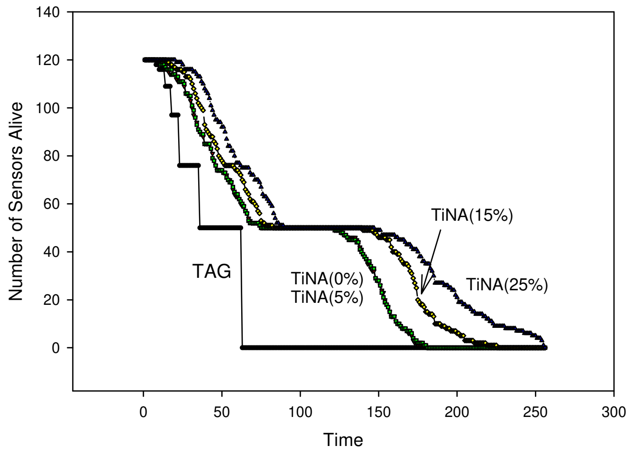

Figure 10: Measurement of lifetime of network

Figure 10 shows our results for this experiment. As our results show, using TiNA can increase the lifetime by 187% in the case of *tct=0* and over 300% in the case of *tct=0.25*. In addition, at any given point in time the simulations using TiNA had at least as many (though usually many more) sensors alive then the base case. This experiment shows that by using TiNA, the sensor network will be able to last for a longer period of time and thus enable queries to last longer. Another finding from this experiment is that using the larger *tcts* has a positive effect on lifetime. There are additional savings of 25% and 42% (versus *tct=0*) for *tct=0.15* and *tct=0.25* respectively. The reasons for extended lifetime are by saving transmission sizes, the sensor nodes are able to transmit less per query period and thus last further into the future (for more periods). Although connectivity is not accounted for in this experiment, the above results strongly indicate that using TiNA, data from more node can reach the base station for longer periods.

6.6 Sensitivity to the Epoch Duration

The final experiment we ran tested the effect of congestion on the proposed scheme. Congestion can occur because the epoch is not large enough to begin with or because network congestion is causing the epoch to no longer be sufficient. This experiment intends to find out how helpful TiNA can be in these special cases. Up to now the default epoch duration has always been large enough to handle all nodes.

Figure 11 shows that by increasing the epoch duration, the energy consumption increases until it levels off. This is because by increasing the epoch duration, more nodes are able to access the channel during the assigned synchronization interval and transmit their readings. However, TiNA is using the available time and energy "wisely". This can be further illustrated by Figure 12. Consider the case where the epoch duration is 280 mSec. TAG required 1.4 times the energy used by TiNA(0%) and TiNA delivered results with QoD 2% higher than TAG. The TiNA sensors send only the readings that changed from the previous epoch. This selectivity in transmission alleviated the network congestion and allowed more valuable data (the readings with changed values) to make its way up the network. On

[2]Increasing the number of bits only further increases the lifetime savings that TiNA creates

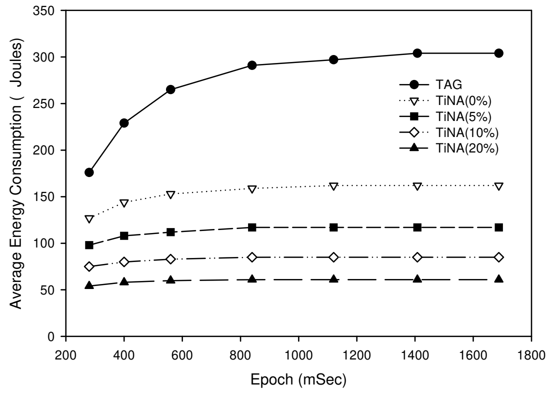

Figure 11: Energy usage vs epoch duration

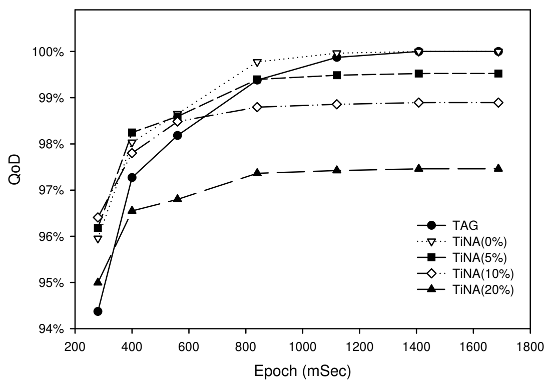

Figure 12: QoD vs epoch duration

the other hand, the TAG sensors send the reading all the time, which resulted in communication congestion, which in turn yielded poor QoD. Moreover, energy is still required to transmit those unchanged readings that managed to cross the congested portions of the network through the non-congested parts. This explains the energy savings provided by TiNA.

7 Conclusions and Future Work

Recent advances in hardware development have enabled the use of wireless sensor networks in a myriad of monitoring applications. Being battery-powered, sensor network designs must be vigilant about energy conservation in order to increase the lifetime of the deployment.

The contribution of this paper is a new scheme for doing temporal in-network aggregation, called TiNA, which balances the trade-off between the quality of results returned to users and energy consumption. TiNA extends current in-network aggregation methods by utilizing temporal coherency tolerance to minimize the size and number of transmitted messages. Since data transmission is the biggest energy-consuming activity in sensor nodes, using TiNA results in signifi cant energy savings.

Our experiments have shown large savings in energy over typical in-network aggregation methods without signifi cant loss in quality of data. TiNA has also been shown to increase quality of data in cases where the period to send is too short and can increase the lifetime of the network. In conclusion, TiNA provides a good trade-off between decreasing energy versus decreasing quality of data in sensor networks. Currently, we are expanding TiNA to consider spatial and topological redundancy.

References

[1] P. Deolasse, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic web data. In *Proc. of WWW10*, May 2001.

[2] S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: Taking lessons from mpeg. In *WWW*, pages 265–274, 2001.

[3] 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.

[4] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-effi cient communication protocol for wireless microsensor networks. In *HICSS*, Jan 2000.

[5] J. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro.*, 22(6):12–24, Nov/Dec 2002.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of MOBICOM*, Aug 2000.

[8] 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*, Oct 2002.

[9] C. Lin, C. Federspiel, and D. Auslander. Multi-sensor single actuator control of HVAC, 2002.

[10] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of OSDI*, 2002.

[11] 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.

[12] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM Intl. Workshop on Wireless Sensor Networks and Applications, (WSNA)*, Sep 2002.

[13] A. Manjeshwar and D. Argrawal. Apteen: A hybrid protocol for effi cient routing and comprehensive information retrieval in wireless sensor networks. In *IPDPS*, 2002.

[14] S.Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proc of ACM SIGMOD*, May 2000.

[15] H. Schwetman. CSIM user's guide. MCC Corporation.

[16] M. A. Sharaf and P. K. Chrysanthis. On-demand broadcast: New challenges and algorithms. In *First Hellenic Data Management Symposium (HDMS)*, 2002.

[17] M. A. Sharaf and P. K. Chrysanthis. Semantic-based delivery of olap summary tables in wireless environments. In *International Conference on Information and Knowledge Management (CIKM)*, 2002.

[18] M. A. Sharaf, Y. Sismanis, A. Labrinidis, P. K. Chrysanthis, and N. Roussopoulos. Effi ecient dissemination of aggregate data over the wireless web. In *Workshop on the Web and Databases (WebDB)*, 2003.

[19] S. Singh and C. Raghavendra. PAMAS: Power aware multi-access protocol with signalling for ad hoc networks. *ACM Computer Comm. Review*, 28(3):5–26, July 1998.

[20] Y. Yao and J. Gehrke. Query processing for sensor net. In *Proc. of CIDR*, Jan 2003.

[21] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, Oct 2002.