TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation [*]

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

ABSTRACT

This paper presents TiNA, a scheme for minimizing energy consumption in sensor networks by exploiting end-user tolerance to temporal coherency. TiNA utilizes temporal coherency tolerances to both reduce the amount of information transmitted by individual nodes (communication cost dominates power usage in sensor networks), and to improve quality of data when not all sensor readings can be propagated up the network within a given time constraint. TiNA was evaluated against a traditional in-network aggregation scheme with respect to power savings as well as the quality of data for aggregate queries. Preliminary results show that TiNA can reduce power consumption by up to 50% without any loss in the quality of data.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Distributed databases, Query processing*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Sensor Network, Power-aware In-network Aggregation

1. INTRODUCTION

Recent advances in hardware development have enabled the creation of widespread sensor networks. Currently, there are many ongoing projects that use wireless sensor networks for environmental monitoring and data acquisition applications. Examples include wildlife tracking [6], habitat monitoring [10], and building monitoring [7]. Sensor nodes, such as the Berkeley MICA Mote [4], are getting smaller, cheaper, and able to perform more complex operations, including having mini operating systems embedded in the sensor. While these advances are improving the capabilities of sensor nodes, there are still many crucial problems with deploying sensor networks. Limited storage, low network bandwidth, poor inter-node communication, limited computational ability, and low power capacity still persist.

In-network aggregation partially addresses the problem of limited power. With in-network aggregation, the sensor nodes carry out part of the aggregation rather than just propagating raw readings. In-network aggregation reduces power consumption because sensor power usage is dominated by transmission costs. Thus, 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.

In this paper we focus on the problem of saving energy as well as maintaining quality of data (*QoD*) in the presence of limited power in sensor networks. We propose a new scheme, called *TiNA* (**T**emporal coherency-aware **i**n-**N**etwork **A**ggregation), which uses temporal coherency tolerances in addition to in-network aggregation to save energy in the sensor network while retaining the user-specified QoD requirement [1]. TiNA is independent of the underlying synchronization protocol used for sending and receiving data between the sensor nodes.

The basic idea behind 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 node.

Our proposed scheme, TiNA, saves energy for two reasons. First, at the edge nodes, a new reading is not transmitted if it falls inside the given tolerance. Secondly, no transmission from the edge nodes further reduces message sizes transmitted by the internal nodes due to groups being eliminated. We will show that by not sending and decreasing the size of messages, the sensor network can gain significant savings in energy from the reduced transmission power requirements while maintaining QoD. We also show that TiNA leads to better QoD when not all sensor readings can be propagated up the network under a given time constraints.

In the next section we present related work. Section 3

[*]This work is supported in part by NSF award ANI-0123705. The first author is supported in part by the Andrew Mellon Predoctoral Fellowship.

provides an overview of sensor networks and, in particular, how TiNA fits into the sensor network. TiNA, our scheme for in-network aggregation, is introduced in Section 4. Section 5 describes our testbed, experiments and results.

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 limited energy. Understanding the events, measures, and tasks required by certain applications has been shown to provide efficient 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 [5].

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 now 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 [12]) 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 root node.

The work on TinyDB [8, 9] and Cougar [16] mapped the directed diffusion concepts in database terms. These projects used an SQL-style query syntax to express a node request for data. Cougar 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 aggregation allows incremental computing of partial aggregates along the different paths of a routing tree to the root, which is the node responsible to collect the results. 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.

A requirement for in-network aggregation is the synchronization between nodes on a single path to the root. A sensor needs to wait before sending its own reading in order to increase the effectiveness of aggregation. While waiting, a sensor receives readings from other neighboring nodes. This coordination allows a routing sensor to compute the aggregate of readings from different sensors 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 will still enable 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 sensors. 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

Queries in sensor networks are continuous, with new results generated periodically. Initially a base station receives a query, which then forwards to the nearest sensor node. This node will be in charge of disseminating the query down the network and collect the results. Aggregation queries received by the base station are in the following form:

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

TAG [8] (the aggregation service for TinyDB) introduced the new clause *EPOCH DURATION i*. Parameter i is the epoch interval and it specifies the arrival rate of new results required by the user. That is, once every epoch, the user is expecting the network to produce a new answer to the posed continuous query. We introduce, as part of the TiNA framework, the clause VALUES WITHIN *tct* in order to express temporal coherency tolerance.

The way in-network aggregation is performed depends on the query *attributes* and *aggregates*. Specifically, 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 sensors 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 is simply the sum of all readings that are forwarded through this sensor. However, if the aggregate function is AVERAGE, then each routing sensor will generate a partial aggregate that consists of the sum of the readings and their count. Eventually, the root sensor 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.

For routing, we use the general directed diffusion paradigm. The purpose is to build a routing tree where each sensor is assigned a *gradient* [3] or a *parent* [8]. A parent sensor for a child sensor c is the sensor through which c will send a message that it wants to propagate up the network. To build the routing tree, each sensor i is assigned a level L_i and a parent P_i. Initially for all the nodes, $L_i = \infty$ and $P_i = 0$. The root node that initiates a query sets its level (L_{root}) value to 0. Nodes exchange query messages to build the routing tree. The header of such a message contains two fields: Id_s and L_s. Id_s is an identifier of the source node that is transmitting the message, while L_s is the level value (i.e., L_i) of that source node. When a child receives a query message and its level is ∞, it sets its parent to the node it heard from and sets its level to $L_s + 1$. The process continues until every node in the network receives a copy of the query message and is assigned a level and a parent value.

As we mentioned in Section 2, synchronizing the transmission between nodes on a singe path to the root is crucial for efficient in-network aggregation. A parent node has to wait until it hears the readings reported from all its children before reporting its own reading. Thus, a parent node p is able to combine the partial aggregates reported by its children with its own reading. Node p 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 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) and the duration of each interval as *(EPOCH DURATION)/d*. During one interval, a parent node will be active and receiving messages from its children. In the next interval, the children nodes will be idle, while the parent is still active transmitting the partial aggregate result. The parent sensor will become idle in the subsequent interval (i.e., when it finished receiving and transmitting the partial aggregates in its subtree).

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

Our proposed scheme will work with any synchronization method. In this paper, we will illustrate TiNA using the TAG approach for synchronization. The advantages of the TAG approach is the guaranteed delivery of a query result every epoch duration and minimizing the energy consumed by a sensor. The latter is because a sensor is required to be active for only two intervals during one epoch: 1) when it is receiving messages from its children, and 2) when it is transmitting the partial result.

4. TINA

Even in-network aggregation, particularly with massive number of sensors, requires a lot of energy to periodically transmit all sensor readings which shortens the lifetime of the sensor 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 at the specified rate.

In TiNA, we exploit the temporal correlation in a sequence of sensor readings to suppress values, reducing energy consumption and at times increasing QoD. TiNA can be tuned to further reduce energy consumption while still providing high quality approximated answers. Since approximation in TiNA is guided by temporal correlation, the degrading in QoD 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 preliminary analysis to identify an interesting trend in the network. Once the trend is identified and higher accuracy is needed, TiNA is tuned to increase QoD 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-network aggregation schemes perform aggregation at the internal nodes as information is passed up the routing tree.

The contribution of TiNA comes in how it decides what data to forward up the routing tree and how to merge or aggregate this forwarded data. TiNA adds a new clause to the query specification called: `VALUES WITHIN tct`. Parameter *tct* is used by the user to specify the temporal coherency tolerance for the query. The value of *tct* specifies the degree to which the user is tolerant to changes in the value of sensor readings. 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.

A TiNA sensor node must keep additional information in order to utilize temporal coherency tolerances. The information kept depends on the sensor's 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 $\frac{|V_{new} - V_{old}|}{V_{old}} > tct$, otherwise the value V_{new} is suppressed.

At an internal (i.e, parent) node, data received from different children is combined to compute the partial result to forward up the routing tree. If a parent does not receive complete data from one of its children, it uses some or all of the last reported data from that child.

The internal node then considers its own reading. If the reading can be aggregated within a group that already exists in the partial result, then the reading is aggregated regardless of its *tct* value. This is because the partial aggregation will only change the partial aggregate value of the group, but it will not increase the size of the partial result. However, if the new reading results 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.

The internal node at this point 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 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. 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 above.

Values of tct higher than 0 can still be applied at the partial aggregate level. This allows for further savings in message passing, however, its semantics from a user perspective

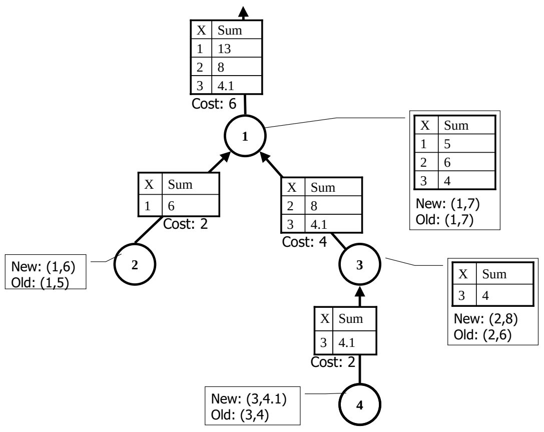

Figure 1: Epoch of query without TiNA

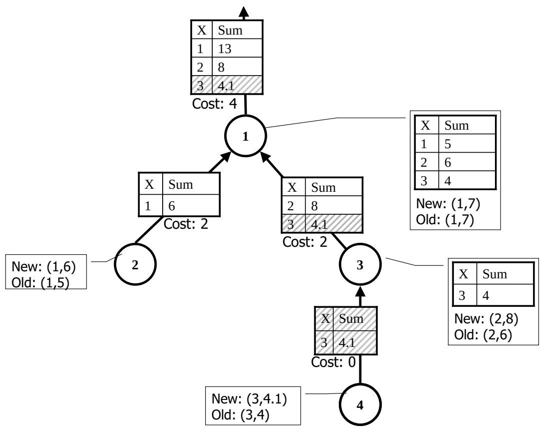

Figure 2: Epoch of query using TiNA

is not as clear as that on the reading level. In Section 5.2 we present an experiment where we used the same *tct* value on both the reading and partial aggregate levels.

A special issue that arises in the context of TiNA is how to handle the case of queries with a WHERE clause. In the in-network aggregation mechanism, if the sensed data does not satisfy the WHERE predicate, then the sensor will suppress transmission. The problem becomes how would the parent determine the difference between a value not satisfying the predicate and a value within the range specified by the *tct*. The way to handle this is by having the child node send a short *invalidation* message to its parent whenever the sensed value does not satisfy the WHERE predicate, thus the parent will not use the stored reading for that child.

A relatively similar issue is how to handle failing sensor nodes where a parent will not able to distinguish between a node failure and the suppression due to applying TiNA. To solve this problem, a child node that is suppressing transmission will periodically send a *heartbeat* message to its parent even if the sensed value is still within the *tct* limit. If the parent node did not receive the heartbeat message for a certain period of time, then it invalidates the entry for that specific child until it hears from it again. This technique will also work for mobile sensors, where a sensor might change its location in the network, thus switching parents.

4.2 Example Execution of TiNA

Figures 1 and 2 show a comparison between two systems during one epoch, one with and one without 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 figures, nodes are represented as circles and the data flow from child to parent is represented with arrows. Tables along the connection lines represent the data that is being sent from child to parent. The boxes connected to each node represent the current state at the node. This current state consists of its last reported reading, called *Old*, its current reading made, called *New*, and a table representing the aggregation of its children old reported data. The Cost number under each of these tables is the cost to send this table from child to parent. The cost is simply 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. See Section 5.1 for the details on how total cost was computed.

In Figure 1, we see the epoch 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 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.

In Figure 2, we see the benefit of using TiNA. The set up is the same as above, but now the sensors 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. For example, when it is time for node 4 to send, it first 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, because of Node 2. In this case, to ensure correctness, Node 1 has to aggregate its new reading into Node 2's 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, significant 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.

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

4.3 Discussion

Exploiting temporal correlation has been used before to suppress the transmission of sensor readings. For example, the *PREMON* paradigm for motion detection uses the spatio-temporal correlation in sensor readings to reduce transmissions [2]. 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. On receiving a prediction model, a sensor will send a new reading only when it differs from the one in the motion prediction model. The *APTEEN* protocol [11], like TiNA, attempts to reduce the number of transmitted readings. APTEEN uses filters to drop values (using a hard threshold) and it exploits temporal correlation to suppress values (using a soft threshold). The values for the hard and soft thresholds are expected to be selected by the application using APTEEN.

Our work is the first to utilize the temporal correlation of sensor readings in the context of in-network aggregation. It provides the mechanisms required to optimize aggregate query processing in sensor network while delivering results of high quality. In addition to the obvious advantages of reducing the energy consumption, TiNA increases QoD in the cases of dense network traffic 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 final aggregate.

5. EXPERIMENTS

5.1 Simulation Environment

To study the effects of the proposed scheme, we created a simulation environment using CSIM [14]. The simulated network consisted of a 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 [15]. 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 [8]. 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 [8], 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.

Parameter	Value	Default
Grid Size	15x15 Cells	
Aggregate	MAX, SUM, AVG	SUM
Number of Attributes	0, 1, 2	1
Epoch Durations	280 – 1700 mSec	1500 mSec
tct	0% – 25%	
Randomness Degree	0.0 – 1.0	0.5
Random Step Size Limit	10% of domain	
Number of Epochs	100 Epochs	

Table 1: Simulation Parameters

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.

Metrics In our experiments we used two measurements: *energy consumption* and *quality of data*. Energy is consumed in four main activities in sensor networks: transmission, listening, processing, and sampling. We focused on transmission power since the amount of time spent listening depends on the synchronization and tree building methods, of which our scheme is independent. We did not include energy required for processing since 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 is dependent on the partial aggregate size and number of messages containing this aggregate.

Below, we report the average energy consumed by a sensor node per epoch. The values of the parameters needed

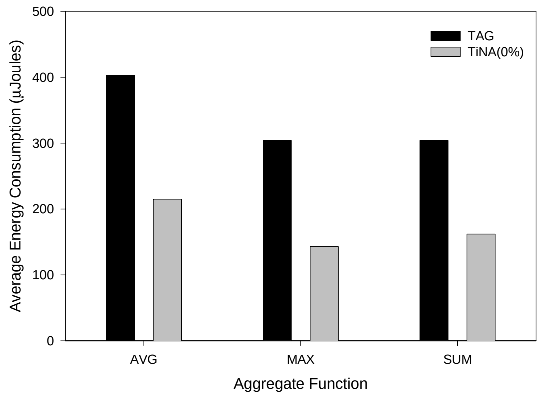

Figure 3: Energy for different aggregate functions

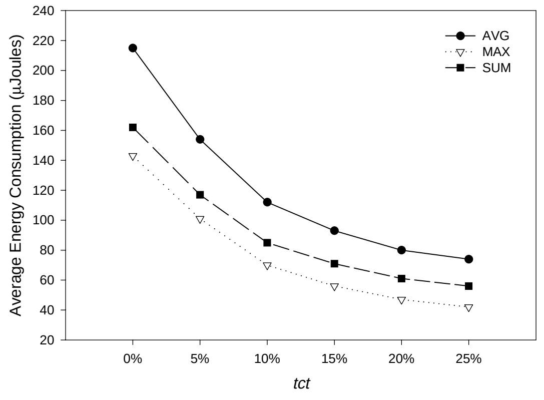

Figure 4: Energy for aggregate functions vs *tct*

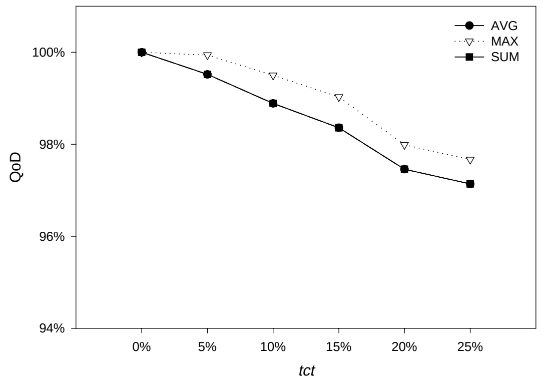

Figure 5: QoD for aggregate functions vs *tct*

to calculate the transmission cost were the same as in [4]. Specifically, we simulated sensors 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.

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 first need to measure the error over the Group-By query. We measured this error as described in [13]. 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., $\epsilon_i = \frac{(|m_i - m_i'|)}{m_i} \times 100$. The error δ over the Group-By query is defined as: $\delta = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i$ Finally, the QoD over time is defined as:

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

where δ_t is the query error at $epoch_t$.

5.2 Experiments and Results

5.2.1 Sensitivity to the Aggregation Function

In the first experiment, we test the sensitivity of our scheme to the different aggregation types. Figure 3 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 3, 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 4 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 5 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

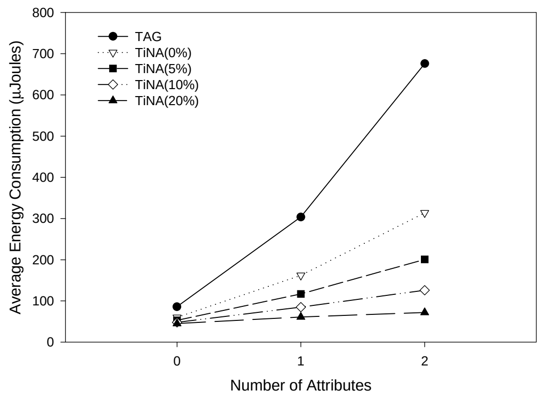

Figure 6: Energy vs number of attributes

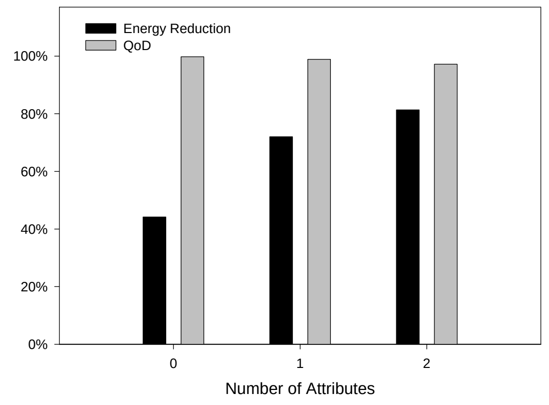

Figure 7: QoD and Energy reduction of TiNA(10%) compared to TAG for different number of attributes

affecting the overall MAX, but in other cases (i.e., AVG and SUM), any change affects the entire result.

5.2.2 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 6 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 benefit 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 7 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.

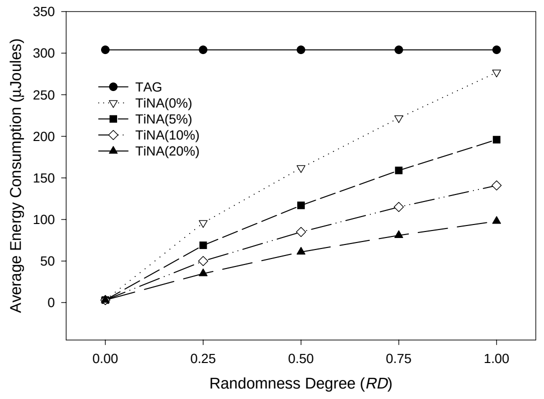

Figure 8: Energy consumption vs Randomness

5.2.3 Sensitivity to the Data Change Rate

The next experiment focuses on the rate at which data changes. Figure 8 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 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%).

5.2.4 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

5.2.5 *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.

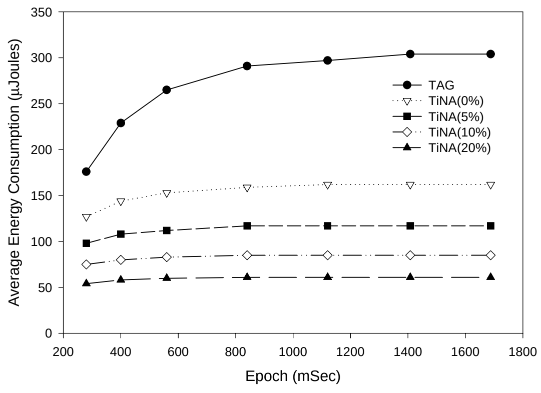

Figure 9: Energy usage vs epoch duration

Figure 9 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 10. Consider the case where

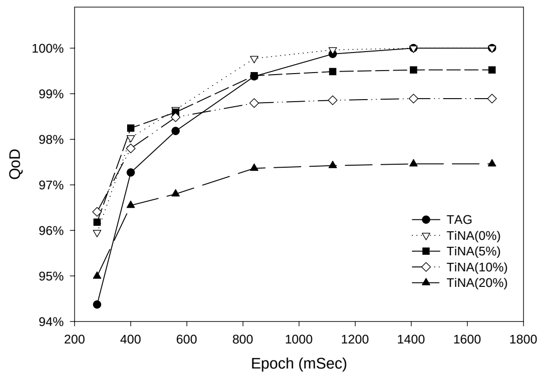

Figure 10: QoD vs epoch duration

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 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.

6. CONCLUSIONS AND FUTURE WORK

We have developed a new scheme for doing temporal in-network aggregation called TiNA. 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 significant energy savings. Our experiments have shown large savings in energy over typical in-network aggregation methods without significant loss in quality of data. 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.

7. REFERENCES

[1] P. Deolasse, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic web data. In *Proc. of WWW*, pages 265–274, 2001.

[2] S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: Taking lessons from MPEG. *Computer Comm. Review*, 31(5), Oct. 2001.

[3] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proc. of SOSP*, Oct. 2001.

[4] J. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro.*, 22(6), 2002.

[5] C. Intanagonwiwat et al. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of MobiCom*, Aug. 2000.

[6] P. Juang et al. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *Proc. of ASPLOS*, Oct. 2002.

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

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

[9] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of SIGMOD*, June 2003.

[10] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of WSNA*, Sep. 2002.

[11] A. Manjeshwar and D. P. Agrawal. APTEEN: A hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks. In *Proc. of IPDPS*, 2002.

[12] C. Perkins. Ad-hoc on demand distance vector routing (AODV). Internet-draft, Nov. 1997.

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

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

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

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